NPS-EC-00-005

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Center for Joint Services Electronic Warfare

An All-Digital Image Synthesizer for Countering High-Resolution Imaging Radars

by

P. E. Pace
S. Ekestorm
C. Karow
D. Fouts

February 24, 2000

Prepared for: Office of Naval Research

NAVAL POSTGRADUATE SCHOOL
Monterey, California

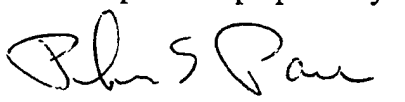
RADM ROBERT C. CHAPLIN
Superintendent

R. Elster
Provost

This report was sponsored by the Office of Naval Research.

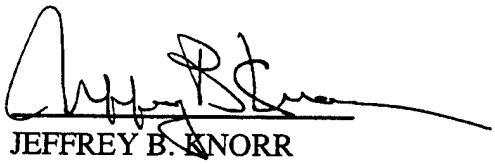
Approved for public release; distribution is unlimited.

The report was prepared by:



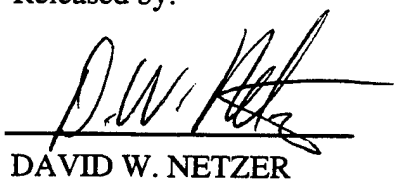
Phillip E. Pace
Associate Professor
Department of Electrical and
Computer Engineering

Reviewed by:



JEFFREY B. KNORR
Chairman
Department of Electrical and
Computer Engineering

Released by:



DAVID W. NETZER
Associate Provost and
Dean of Research

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	February 24, 2000	Technical Report

4. TITLE AND SUBTITLE
An All-Digital Image Synthesizer for Countering High Resolution Imaging Radars

5. FUNDING NUMBERS
N00173-00-WR 0014

6. AUTHOR(S)
P.E. Pace, S. Ekestorm, C. Karow, and D. Fouts

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Department of Electrical and Computer Engineering
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER
NPS-EC-00-005

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Office of Naval Research
Code ONR-313EW
800 North Quincy Street
Arlington, VA 22217-5660

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this report are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE
A

13. ABSTRACT (Maximum 200 words)

A digital image synthesizer (DIS), especially useful as a counter-targeting signal repeater, (i.e, for synthesizing the characteristic echo signature of a pre-selected target) is reported. The DIS has a digital radio frequency memory (DRFM) and associated circuitry, including digital tapped delay lines and a modulator in each delay line to impose both amplitude and frequency modulation in each line. A unique property of the digital image synthesizer is its ability to synthesize false targets using wideband chirp signals of any duration. To generate the target, the user can program the target extent (number of taps) and the amplitude and Doppler frequency of each range-Doppler cell within the image. The *system-on-a-chip* uses a scalable CMOS technology that increases the bandwidth and sensitivity of such a repeater over prior analog based systems. The application specific integrated circuit reduces the noise of the repeated signal, reduces the size and cost of such a system, and permits real time alteration of operating parameters, permitting rapid and adaptive shifting among different types of targets to be synthesized. A scan path test capability is also included to allow intra-chip signal analysis and verification.

14. SUBJECT TERMS	15. NUMBER OF PAGES
digital radio frequency memory, DRFM, counter-targeting, image synthesizer, application specific integrated circuit, ASIC, electronic warfare	268
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR

EXECUTIVE SUMMARY

A digital image synthesizer (DIS), especially useful as a counter-targeting signal repeater (i.e., for synthesizing the characteristic echo signature of a pre-selected target) is reported. The DIS has a digital radio frequency memory (DRFM) and associated circuitry, including digital tapped delay lines and a modulator in each delay line to impose both amplitude and frequency modulation in each line. A unique property of the digital image synthesizer is its ability to synthesize false targets using wideband chirp signals of any duration. To generate the target, the user can program the target extent (number of taps) and the amplitude and Doppler frequency of each range-Doppler cell within the image. The *system-on-a-chip* uses a scalable CMOS technology that increases the bandwidth and sensitivity of such a repeater over prior analog based systems. The application specific integrated circuit reduces the noise of the repeated signal, reduces the size and cost of such a system and permits real time alteration of operating parameters, permitting rapid and adaptive shifting among different types of targets to be synthesized. A scan path test capability is also included to allow intra-chip signal analysis and verification.

1. COUNTERING THE SENSOR-SHOOTER ENGAGEMENT

Future Navy electronic warfare (EW) systems must be designed to operate in the RF environment to provide a layered EW defense and also serve as a fully integrated shipboard combat system sensor. Next generation EW systems must also provide threat identification and a complete situational awareness to allow the quick reaction modes required to counter the modern anti-ship cruise missile (ASCM) threat. Figure 1.1 shows the sequence of events taken by the enemy sensor-shooter in order to place a missile on a target (hard kill). A typical sequence begins with the enemy's electronic support surveillance sensor detecting the target of interest (e.g., with a long range over-the-horizon targeting radar). After acquiring a number of hits on the target, the identification of the target is pursued using an additional high-resolution sensor such as an airborne inverse synthetic aperture radar (ISAR) imager. This type of radio frequency (RF) sensor forms an image of the target that can be used for recognition and identification.

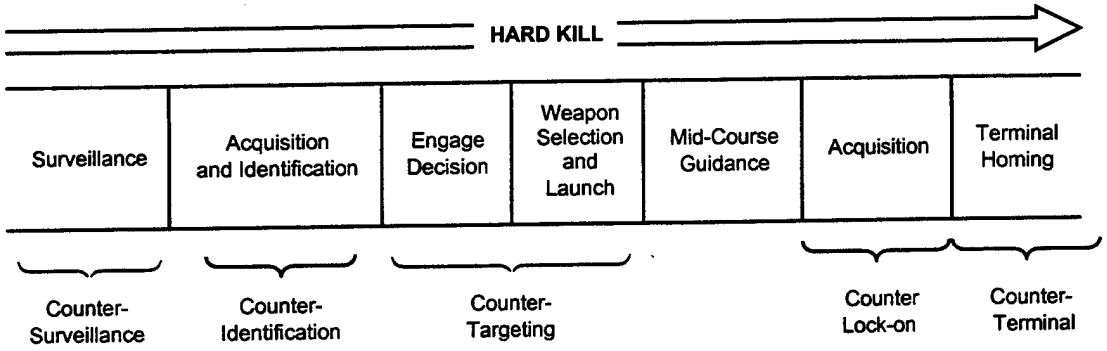


Figure 1.1: Sequence of steps necessary to land a missile on a target.

Depending on the target identification, the decision to engage the target and launch a weapon (such as an ASCM) is made using the inputs, for example, from the ISAR imager. After the ASCM is launched, acquisition and terminal homing of the

missile is again accomplished using the missile's ISAR. Use of an ISAR in the terminal phase of the missile allows a good aimpoint accuracy and greater probability of kill.

To avoid the ASCM hard kill, a number of countering techniques can be used including counter-surveillance, counter-identification, counter-targeting, counter-lock-on and counter-terminal. Counter-surveillance and counter-identification include the use of low radar cross section materials, stealth and deception devices. Counter-targeting includes the use of active electronic attack (EA) and the use of decoy repeaters. Counter-lock-on and counter-terminal techniques consist of EA, distraction and seduction chaff as well as the use of decoy repeaters.

Counter-identification and counter-targeting systems can begin the electronic attack well before the opposition launches any missiles due to the generation of a lower probability of target acquisition. Since acquisition systems and future missile seekers will employ pulse-to-pulse spread spectrum using unfocused SAR and ISAR to improve target recognition and decoy rejection, the need for coherent countering of these imaging sensors/seekers remains a high priority for EA systems. Countering-identification and counter-targeting techniques employ a false target image generated or synthesized with the objective of deceiving the imaging radar into believing the false target is a real one. Imaging sensors use coherent range-Doppler processing and consequently, various forms of complex modulations must be imposed on the intercepted wideband waveforms in order to enable the imager to integrate the false target properly.

In this report, the design, analysis and fabrication of an all-digital image synthesizer for pulse-to-pulse countering of high resolution RF imaging sensors (e.g., SAR, ISAR) is presented. The signal processing used in the digital image synthesizer circuit is especially useful as a signal repeater i.e., for synthesizing the characteristic echo signature of a pre-selected target. The entire system has a digital radio frequency memory (DRFM) and associated circuitry, including a digital tapped delay line and a modulator in each delay line to impose both amplitude and frequency modulation in each range cell. Use of digital semiconductor technology (0.5/0.35 μm CMOS) increases the bandwidth and sensitivity of the repeater over prior analog based systems and reduces the noise of the repeated signal. It also reduces the size and cost of such a system and permits real-time alteration of operating parameters, permitting rapid and adaptive shifting among

different kinds of targets to be synthesized. The integrated circuit is designed such that it can easily be integrated with a number of phase-sampling DRFM architectures.

For completeness, Chapter 2 provides a brief introduction to ISAR and ISAR signal processing. Chapter 3 discusses the digital image synthesizer concept and how the false target in generated. In order to maintain a simulation of the hardware for easy evaluation of concept alternatives, Chapter 4 describes a modular Matlab program that is easy to use and maintain. Chapter 5 presents an Altera field programmable gate array (FPGA) implementation of the image synthesizer concept. To increase the bandwidth of the device, Chapter 6 describes the investigation into converting the FPGA design into an application specific integrated circuit (ASIC). In chapter 7, the ASIC design in scalable CMOS is described in detail including a full-simulation of a 2-tap device. Comparison of the results with the Matlab simulation is also presented in order to verify the concept and detail the advantages of the architecture.

2. INTRODUCTION TO INVERSE SYNTHETIC APERTURE RADAR

ISAR is a high-resolution technique for imaging isolated moving targets such as ships and aircraft. The technique used by both targeting sensors and ASCMs, closely parallels the SAR imaging approach in which the image (or map) is generated from the return signals being reflected off the target as the radar moves past the target area. For the ISAR technique the target imaging is generated from the return signals begin reflected off the target as the *target rotates* within the radar illumination. To understand this duality Figure 1.2 shows a spotlight SAR in which the radar transverses a circular path about the target while collecting the return signals (focused spotlight) [1]. The radar antenna in the spotlight SAR continually tracks the target. Note that the same signal returns could be collected if the radar were stationary and the target was put through a rotation as shown in Figure 2.1 (b).

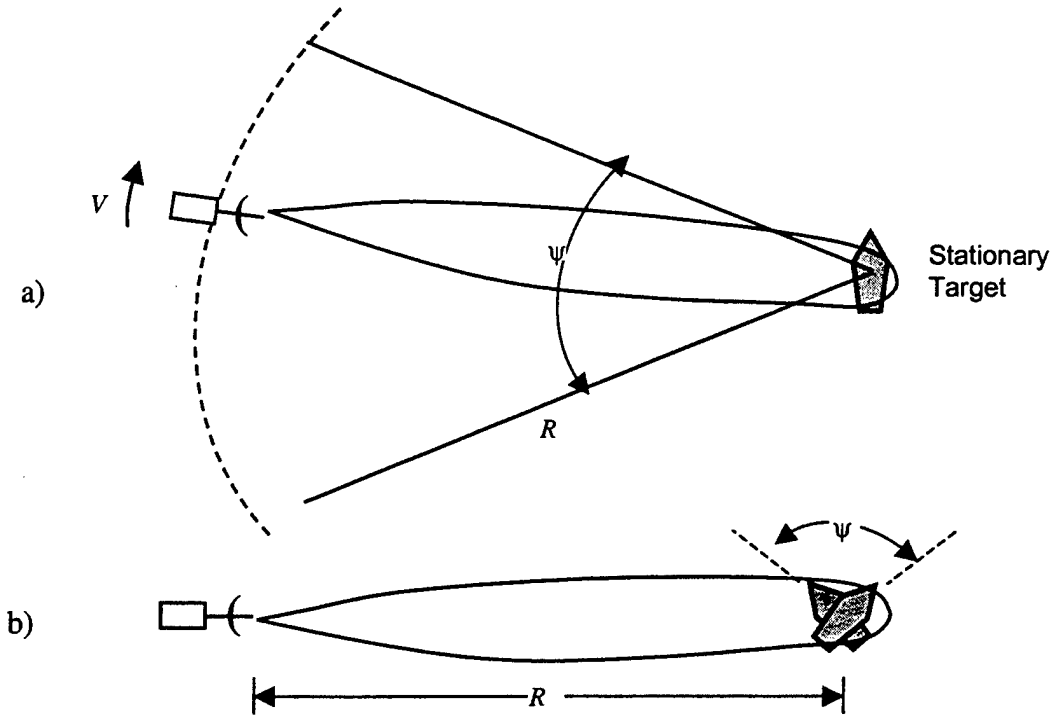


Figure 2.1: Comparison of the geometrical relationship between (a) focused spotlight SAR and (b) ISAR (adapted from [1]).

A. Range Doppler Imaging

The range-Doppler image consists of resolution cells each containing estimates of the target's magnitude and position of scatterers in both range and cross range (Doppler). The orientation of the range-Doppler image is determined by the target's rotation relative to the ISAR. The range dimension within the range-Doppler image is oriented along the radar line of sight (LOS). Range focusing is based on the range-independent point target response determined by the wideband chirp waveform. The cross range dimension of the range-Doppler image is the dimension lying *perpendicular* to the plane contained by the radar LOS and contains the Doppler frequency of the resolved scatterers in range. The azimuth focusing is accomplished by determining the rotational motion during data collection and calculating the compressions for the sharpest focus. The Doppler frequency shift produced by a range resolved scatterer is proportional to the angular rotation rate ω and the cross range distance between the scatterer and the center of target rotation [1].

B. Range Compression Process

High range resolution ISAR uses an analog frequency coding technique, called chirp. A chirp pulse waveform is shown in Figure 2.2. The transmitted chirp can be expressed as a complex narrowband signal

$$S_t(t) = a(t)e^{j\Phi(t)} = \text{rect}\left(\frac{t}{T}\right)e^{j2\pi\left(f_c t + Kt^2/2\right)} \tag{2.1}$$

where f_c is the carrier frequency, Δ is the linear frequency sweep or bandwidth of the transmitted signal, K is the slope or chirp rate ($K = \Delta/T$), T is the pulse width and the

$$\text{rect}\left(\frac{t}{T}\right) = \begin{cases} 1 & \text{for } \left|\frac{t}{T}\right| < \frac{1}{2} \\ 0 & \text{for } \left|\frac{t}{T}\right| > \frac{1}{2} \end{cases} \tag{2.2}$$

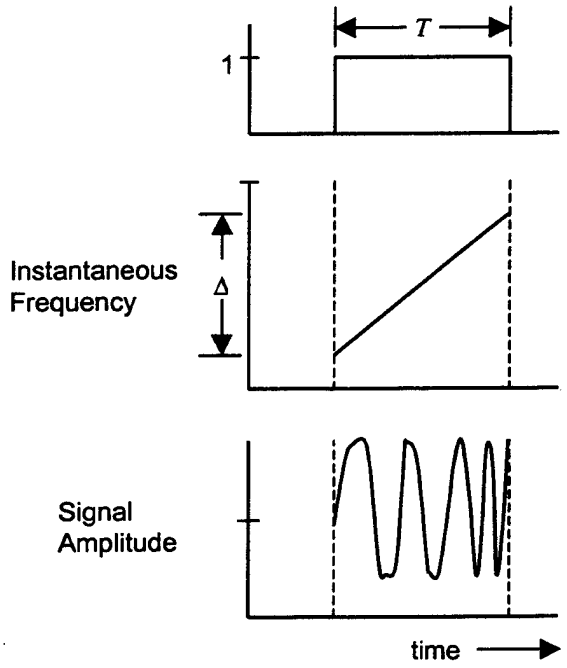


Figure 2.2: Chirp pulse waveform.

instantaneous frequency (time-dependent frequency) is obtained as

$$f(t) = \frac{1}{2\pi}\frac{d\Phi}{dt} = f_c + kt\,. \tag{2.3}$$

Within the pulse duration T, the instantaneous frequency changes from $f_c - kT/2$ to $f_c + kT/2$. The dispersion D or time-bandwidth product of the waveform is $D = T\Delta$ [1].

C. Analog Range Compression Network Example

The chirp pulse waveform can be compressed using an analog pulse compression network as shown in Figure 2.3.

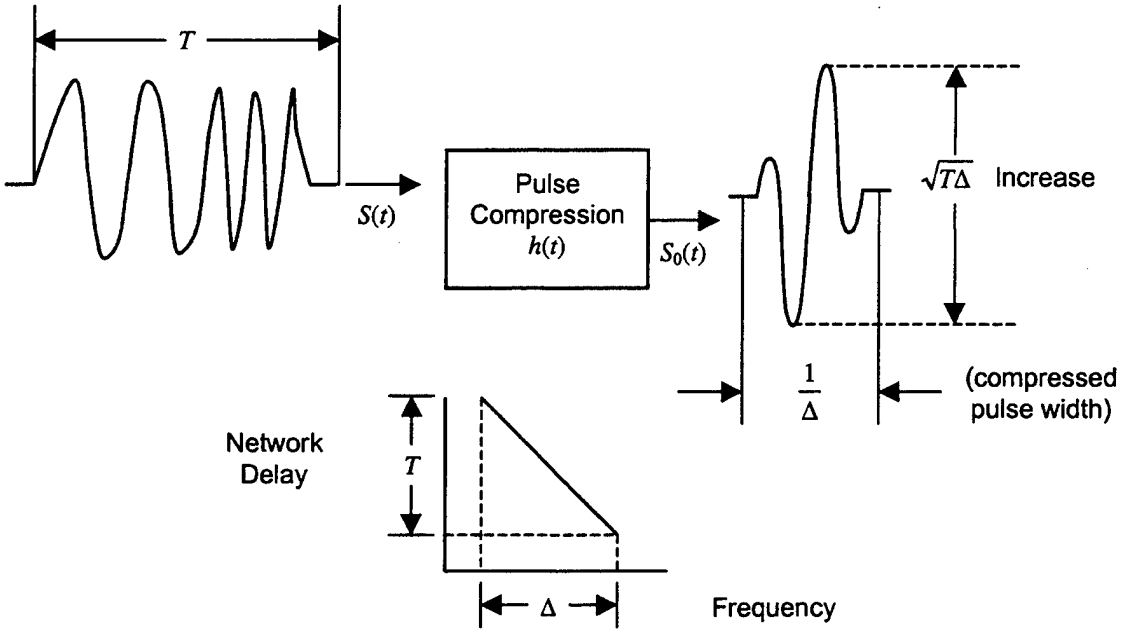


Figure 2.3: Chirp pulse waveform compressed using analog pulse compression network.

This common form of a pulse compression network is called a *phase equalizer* and equalizes the slope of the linear frequency sweep. The transfer function of the pulse compression network can be written as

$$H(f) = e^{j2\pi/K(f-f_c)^2} \tag{2.4}$$

The corresponding impulse response can be expressed as

$$h(t) = \int_{-\infty}^{\infty} H(f) e^{j2\pi ft}\, df \tag{2.5}$$

or

$$h(t) = \sqrt{\frac{j\Delta}{T}}\, e^{j2\pi(f_c t - Kt^2/2)} \tag{2.6}$$

The complex matched filter output is obtained by convolving the chirp signal with the impulse response as

$$S_0(t) = h(t) * S(t) = \sqrt{Dj}\, \frac{\sin(\pi\Delta t)}{\pi\Delta t}\, e^{j2\pi(f_c t - Kt^2/2)} \tag{2.7}$$

The compressed pulse duration of the envelope at the $2/\pi$ points is $T_c = 1/\Delta$ (Raleigh resolution). The corresponding range resolution is then

$$dr = \frac{c}{2\Delta} \tag{2.8}$$

Note the wider the bandwidth of the ISAR chirp signal transmitted, the smaller the range bin size.

D. Digital Range Compression

If the pulse compression is performed digitally on the baseband return samples, the possibility exists to adaptively control the matched filter transfer function. The range resolution is determined by the ADC sampling rate. The convolution can be carried out in the frequency domain using the advantages of the fast Fourier transform (FFT) as

$$S_0(f) = F\{S(t) * h(t)\} = S(f)H(f) \tag{2.9}$$

and is the time domain convolution carried out by multiplication in the frequency domain where $S(f)$ is the spectrum of the returns from one transmitted pulse and $H(f)$ is the transfer function (reference function) of the pulse compression filter which is stored as a series of complex pairs (constant for a particular chirp waveform). The range compression signal processing is shown in Figure 2.4.

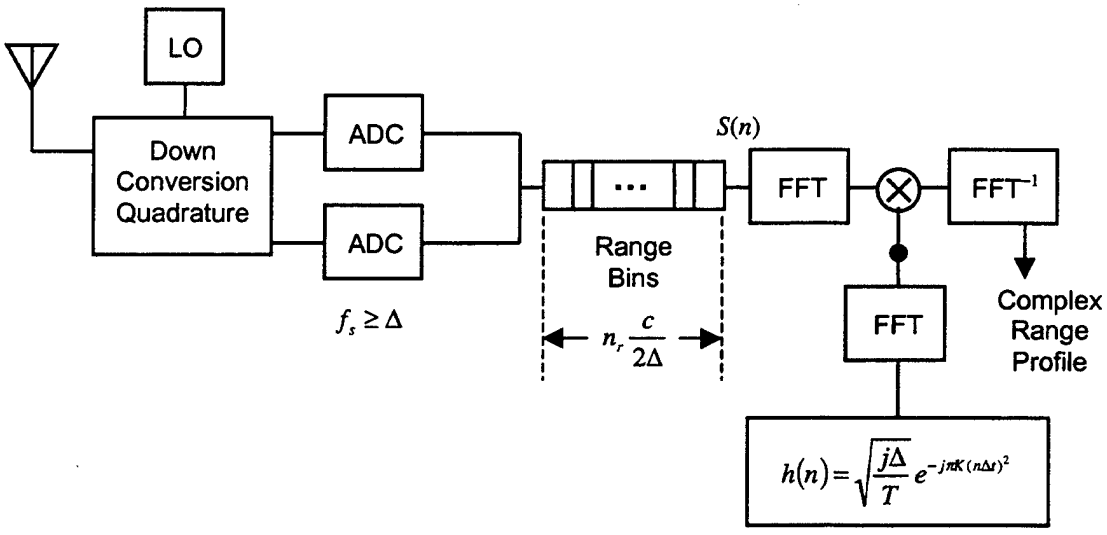


Figure 2.4: ISAR range compression signal processing.

The number of samples required for both $S(n)$ and $h(n)$ to avoid a circular convolution is

$$N \geq \frac{T + \frac{2(R_2 - R_1)}{c}}{\Delta t} - 1 \tag{2.10}$$

where R_2 and R_1 are the edges of the range window to be processed and $\Delta t = 1/f_s$ is the ADC sampling period. Zeroes must be added to the signal and to the $T/\Delta t$ samples of the impulse response (common period of length N). Also note that $N = 2^{\alpha}$ (where α is an integer) due to the constraint on the FFT algorithm. The unambiguous range extent of the ISAR is

$$R_u = \frac{n_r c}{2\Delta} = \frac{n_r c}{2 f_s} \tag{2.11}$$

and depends on the bandwidth of the chirp signal. A two dimensional high resolution spectral analysis algorithm based on 2-D linear prediction using autoregressive estimation for ISAR has been presented in [2]. This approach is superior to the FFT method mentioned above.

E. Azimuth Compression Process

If the target rotates at a rate of ω rad/s towards the radar, a scatterer at a cross range distance a has an instantaneous velocity ωa toward the radar with a corresponding Doppler frequency shift

$$f_d = \frac{2\omega a}{\lambda} \tag{2.12}$$

Considering two scatterers in the same slant range cell separated by da then

$$df_d = \frac{2\omega da}{\lambda} \tag{2.13}$$

resulting in a cross range resolution of

$$da = \frac{\lambda}{2\omega} df_d \tag{2.14}$$

The Doppler resolution is related to the inverse synthetic integration (frame) time $df_d = \frac{1}{T}$ giving a cross range resolution of (see Figure 2.5)

$$da = \frac{\lambda}{2\omega T} = \frac{\lambda}{2\psi}. \tag{2.15}$$

A cross range profile exists for each range bin. Samples that are integrated to form a cross range profile come from the same range bin separated by a pulse repetition interval (PRI) as shown in Figure 2.5.

The unambiguous cross range extent corresponds to the *target size* in the cross range. The required PRF for unambiguous sampling a target of cross range extent A_u is

$$PRF = \frac{2\omega A_u}{\lambda} \tag{2.16}$$

and the number of range samples needed is

$$n_a = \frac{2\omega A_u T}{\lambda} \tag{2.17}$$

The cross range extent is

$$A_u = n_a da = \frac{n_a \lambda}{2\psi} \tag{2.18}$$

A summary of the ISAR compression process is shown in Figure 2.6.

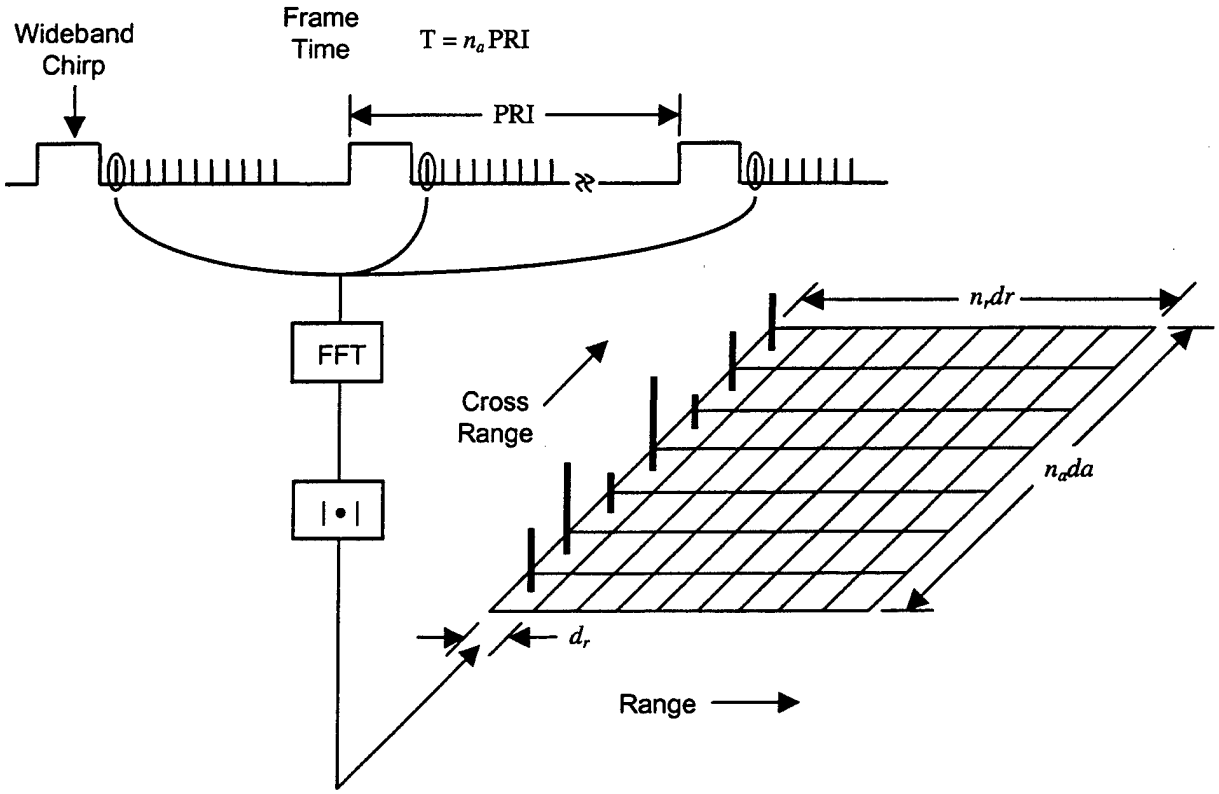


Figure 2.5: ISAR azimuth compression processing.

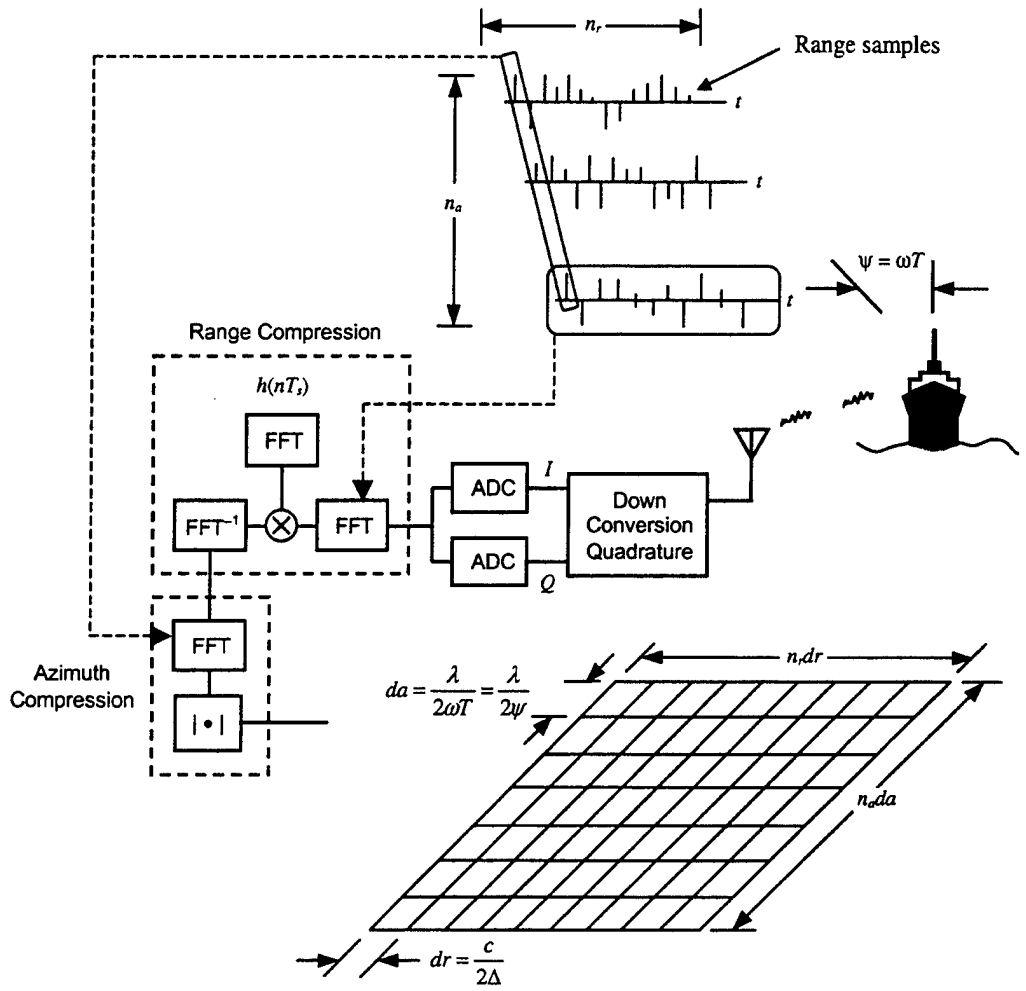


Figure 2.6: Summary of ISAR compression processing.

3. THE DIGITAL IMAGE SYNTHESIZER CONCEPT

A. Scattering Physics of a Target

An object will modify any signal reflected from it according to the object's shape, surface material properties, and the object's velocity relative to the signal. This permits an enemy sensor to identify the nature of such objects, which, if the objects are military platforms like warships or aircraft, is not desirable. One solution presented in Chapter 1 has been to artificially synthesize fake characteristic echo signatures in response to receipt of an interrogating signal. Figure 3.1 shows a ship and an aircraft, in the line of sight of an interrogating radar signal. As the signal hits the aircraft and the ship, it is reflected from their major scattering surfaces. The return signal from the ship and the aircraft will be the superposition of the reflections from the various surfaces such as the hull, superstructure, the aircraft wings and nose. Since these surfaces are at different places along the line of sight to the radar, the superimposed reflections will be out of phase with one another by the differing times of signal propagation to each reflecting surface. This tends to lengthen the return radar pulse by an amount equal to the round trip propagation time of the radar signal between the nearest and farthest major reflector and to make the reflection magnitude vary as dictated by the varying radar cross sections of the reflecting surfaces. Furthermore, movement of the aircraft or ship relative to the radar signal will Doppler shift the returned reflections. That is, any platform which reflects the radar signal will frequency modulate the signal, such that the returned reflections permit the radar to calculate the nature and motion of the platform.

The most common way to detect a Doppler spectrum in the return signal is to compare the reflections from consecutive pulses. Thus, an imaging sensor, such as a search radar, SAR, or ISAR can calculate the Doppler by comparing consecutive return pulses on a range bin by range bin basis. The Doppler spectrum is conventionally computed using an algorithm that incorporates the discrete Fourier transform.

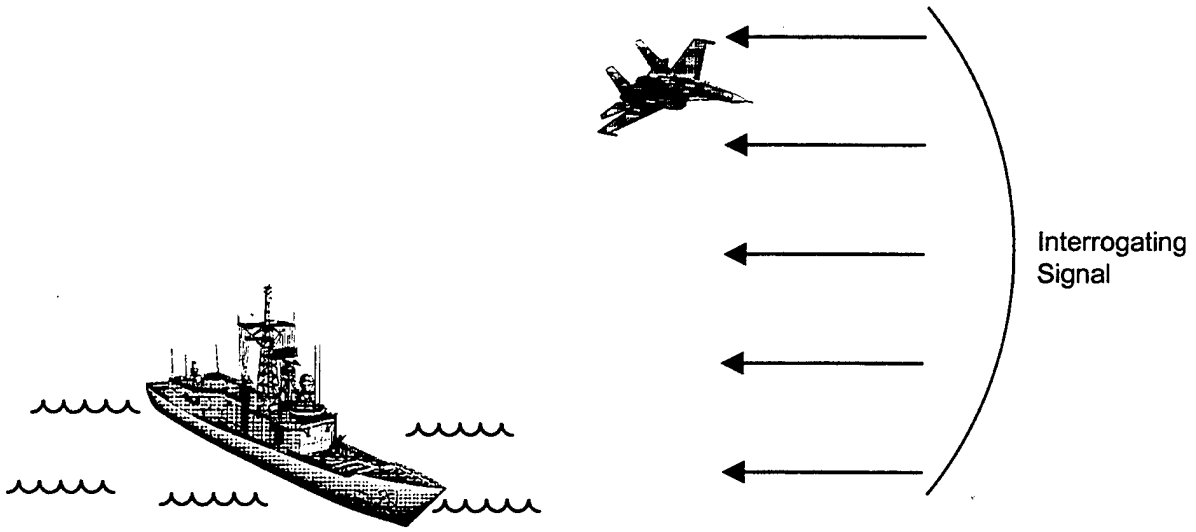


Figure 3.1: A ship and an aircraft in the line of sight of an interrogating radar signal.

B. Analog Image Synthesis

Any credible counter-targeting repeater decoy must synthesize the temporal lengthening and amplitude modulation caused by the many recessed and reflective surfaces, and generate a realistic Doppler shift for each surface. Conventionally this has been done using analog systems that receive an interrogating signal and pass it through a length of cable having serial taps along its length, one tap per range bin. Each tap modulates the signal in amplitude and/or frequency to synthesize the reflection from the reflective surfaces within that range bin. The delay time between taps is selected to correspond to the differing times of flight of the radar pulse to the respective range bins. Finally, the signals from the taps are summed, and the synthesized signal is retransmitted. In this manner, the system returns what appears to be an echo from an object located within the selected range bins having a signature indicative of the moving ship or aircraft object to being synthesized.

Unfortunately, analog systems have drawbacks that limit their usefulness as image synthesizers. They are inherently noisy, and can hold an incoming signal only a short time for processing before the signal deteriorates below the noise. This limits the system bandwidth, and permits effective synthesis of only small objects. Further, analog systems are costly and very bulky, the latter being a particular concern for military platforms, where space is extremely limited. Finally,

analog systems cannot readily change operating parameters such as relative delays among taps, or the amount of modulation in the various taps. This means that analog image synthesizers cannot switch among different simulated objects on the fly, but rather must typically be fabricated for one specific type of target.

C. Digital Image Synthesis

The main advantage of the all-digital image synthesizer repeater is the increase in bandwidth provided to the tapped delay line processors of the kind above described. In addition, the capability to hold the received signals as long as necessary for a given application is provided. Due to the all-digital architecture, modulation of the target extent (number of range bins) and Doppler frequency of each resolution cell is also a capability. This results in a small, low-cost and flexible counter-targeting repeater decoy processor.

The digital image synthesizer uses a DRFM, and an associated digital processing circuit having a plurality of tapped delay lines, a summer in order to sum the output of the delay lines, and range bin signal modulator in each of the delay lines. A DRFM is a semiconductor device that can rapidly and permanently record radio frequency information as digitized samples of the incoming signal, and read it back equally rapidly when needed. Because the DRFM can hold data indefinitely, the duration of the synthesized signal is not limited, as with analog systems, thus permitting (as in the example of Figure 3.1) simulation of larger objects by adding more taps to accommodate more range bins. Because the associated circuitry is digital, and most especially because the circuitry can be dedicated to its processing task (rather than requiring extensive programming to perform its tasks), the speed of the synthesizer can be especially great.

In an optimum hardware configuration, the associated digital image synthesizer circuitry is made part of the DRFM on the same monolithic chip in order to increase the synthesizer speed even more. This is in contrast to a computer, or programmable processor, which, in conjunction with a fast and permanent memory like a DRFM, could in principle do the necessary processing. But the time needed to execute the large number of programming instructions necessary to process data makes this far less desirable than the current design described in this report, and, for

the specific problem of counter-targeting decoy repeaters, largely ineffective.

D. Functional Description of the Digital Image Synthesizer

Figure 3.2 shows a block diagram of the digital image synthesizer [3]. The antenna receives the radar pulse from a (possibly hostile) search radar. After down conversion (not shown), a set of comparators digitizes the phase of the analog signal producing a stream of digital samples which are stored in the DRFM. The phase samples are a digital representation of the phase only. Phase sampling DRFMs have fewer number of comparators and permit coherent reconstruction of the original signal using stored amplitude information [4]. The digitized samples are read serially from the DRFM via the tapped delay line. The circuit of Figure 3.2 shows two taps, but this is illustrative and in principle the device contains the largest number of taps that the particular application dictates (the number of major reflective surfaces of the synthesized target).

The digital phase samples from the DRFM are sequentially read into the taps by clocking. The signals in the respective taps are delayed with respect to one another by pre-selected amounts dictated by the delays. For simplicity, the following discussion references the first tap leg only. However, the function of each leg is identical. The phase signals in the tap, pass through a phase accumulator and an associated look-up-table (contains sine and cosine values for a 2π cycle used in constructing the I & Q components). Although the tap process could readily calculate cos (ϕ_n) and sin (ϕ_n), doing so is less computationally efficient than use of look-up-table, and thus would reduce overall system speed. At the output of the look-up-table, a selectable gain multiplies the signal by a pre-selected amount. Together, these blocks constitute a range bin signal modulator.

The accumulator frequency modulates the signal traversing the tap leg by phase rotation (serrodyne modulation). The phase ϕ of any signal subjected to a linear frequency modulation such as Doppler shift is given by $\phi = (\omega+\omega_d)t$, where ω is signal angular frequency, ω_d is the change in frequency due to the modulation, and t is time. Thus at each point in time the difference in phase between the modulated and unmodulated signal is $\omega_d t$. For a digitally sampled signal, the phase of the nth sample $\phi_n = n(\omega+\omega_d)PRI$, where n is an integer counter and PRI is the period at which the signal is sampled. The phase difference due to the Doppler frequency is $n\omega_d PRI$. Thus one can shift the frequency of a digitally sampled signal by an amount ω_d by rotating each nth phase sample by $n\omega_d PRI$. That is, the frequency of a digitally sampled signal can be shifted by incrementing the phase $n\omega PRI$ of each nth sample by $n\omega_d PRI$.

In summary, the Doppler of a target is typically inferred by sampling target-echoes (within a single range bin) at the pulse repetition rate and inspecting these samples for Doppler

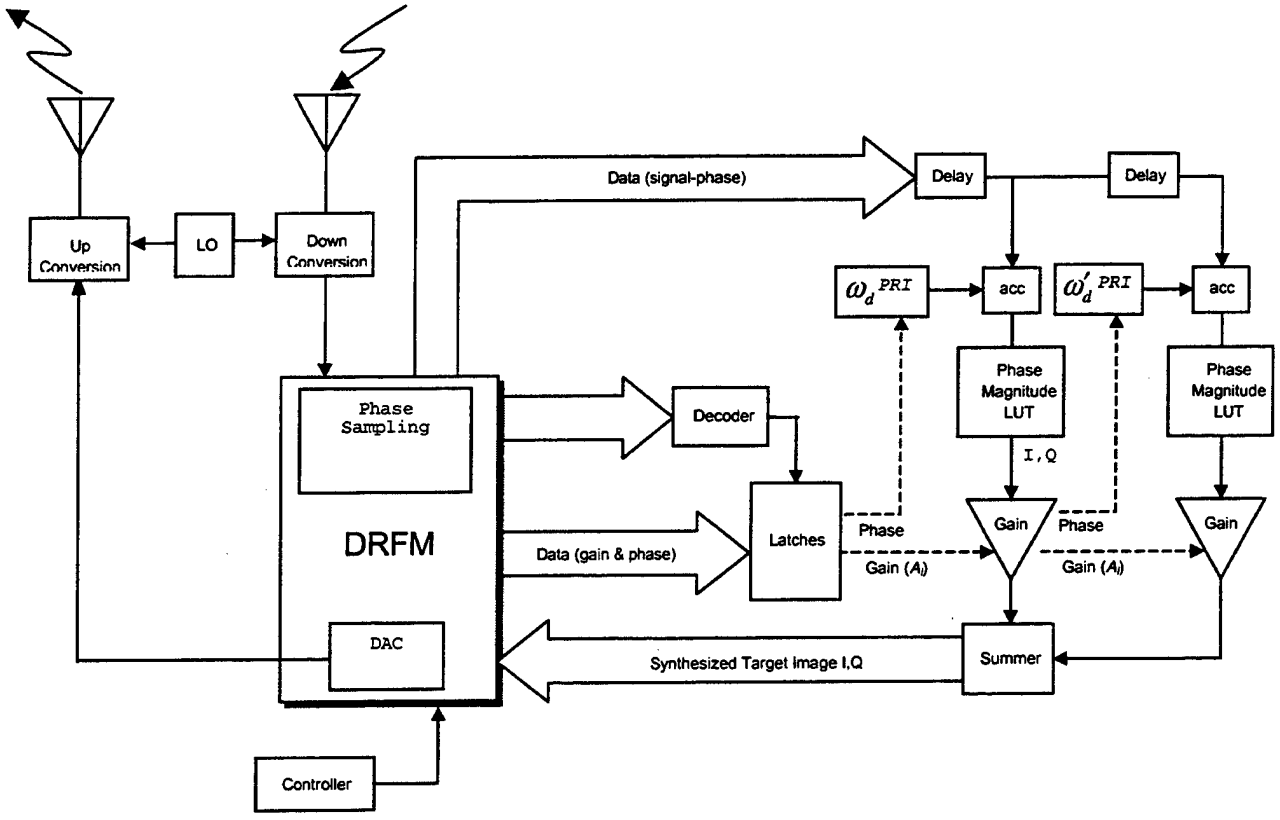


Figure 3.2: Block diagram of the digital image synthesizer (DIS) (adapted from [3]).

induced phase differences between the echoes. One can simulate a Doppler shift of ω_d by repeating the pulses from a sensor, with each pulse phase shifted with respect to the next by an amount $\omega_d PRI$, where *PRI* is the pulse repetition interval. A unique property of the DIS is its ability to synthesize false targets using chirp signals of any duration. The number of tap stages is equal to the target range-extent desired for synthesis.

In operation, the phase accumulator sets nominal values of ω_d and ω_d' per instructions from the DRFM controller. A sensor sends a burst of N pulses having a pulse repetition period of *PRI*. The phase samples from the first pulse (stored in the DRFM) are piped to the first tap leg and the accumulator rotates the phase *of each sample* by an amount $\omega_d PRI$. The resultant phase samples are converted to I and Q components and scaled by a gain factor A_i. In the absence of output from the second tap leg shown, the complex signal is returned to the DRFM, and thereafter to the digital-to-analog converter that reconstructs the analog pulse for up conversion and retransmission.

The waveform of the retransmitted pulse is identical to that of the received pulse, except that it is phase rotated by $\omega_d PRI$. After processing this pulse, the DRFM changes the phase of the first tap accumulator to $2\omega_d PRI$, rotates each phase sample of the *second pulse* by $2\omega_d PRI$, and, again assuming no output from the second tap, retransmits the reconstructed pulse. This continues through the N pulses of the burst, with the phase samples of each pulse rotated by an amount $n\omega_d PRI$, where n is pulse number, i.e., $n = 1, 2, \ldots, N$. In the absence of output from the second tap, the result is a stream of analog pulses from the antenna that are different in phase *from one pulse to the next* by $\omega_d PRI$. A sensor detecting these echoes would interpret the constant pulse-to-pulse phase shift of $\omega_d PRI$ as a Doppler shift from a single reflector. The second tap leg does the same thing, by use of a different ω_d. The summer then combines the output of the first and second tap legs. The complex signal that the summer returns to the DRFM is the superposition of the signals exiting the first and second tap legs. This means that for each n^{th} pulse of the N pulses, the summer's output will be the superposition of two copies of the n^{th} pulse, delayed with respect to one another by the tap delay, scaled differently by the gains A_i, with one phase rotated by $n\omega_d PRI$, the other by $n\omega'_d PRI$. A sensor which receives the corresponding N analog pulses will interpret this as having come from two reflectors located in range bins separated by the delay with reflective cross sections

respectively proportional to the two gains. Because the pulse to pulse phase difference between these pulses is $\omega_d PRI$ for the range bin corresponding to the first delay and $\omega'_d PRI$ for the bin corresponding to the second delay, the sensor will interpret that the reflectors in these two range bins have Doppler frequencies of ω_d and ω'_d, respectively.

The decoder and latch shown in Figure 3.2 updates the phase rotation and gain coefficients for the tap legs. The controller is a process computer interfaced with the DRFM that permits an operator to change these parameters on the fly in real time. In addition to the phase and gain coefficients, the number of taps utilized (target extent) can be changed. Alternatively, the controller can do this automatically. This is particularly important if ω_d in any tap leg varies with time. In the example of Figure 3.1, the aircraft flies directly at the sensor at a constant speed Doppler shifts the signal by a constant, positive, amount. The ship, on the other hand, could be rocking back and forth in the water along the line of sight and thus the Doppler shift corresponding to this motion would oscillate in time.

4. ARCHITECTURE VARIATIONS AND SIMULATION

A. Architecture Variations

Two different implementations of the DIS architecture have been studied. The major difference between the two implementations is the placement of the time-delay processor. Advantages and disadvantages of the two approaches are addressed in this Chapter and are mainly the result of the hardware technologies used. The two different implementations are referred to as the "original architecture" and the "modified architecture".

The "original architecture" described in Chapter 3, is illustrated in the block diagram shown in Figure 4.1. The intercepted chirp signal within the DRFM operating bandwidth is down converted into its I,Q components with a corresponding intermediate frequency that lies within the instantaneous bandwidth of the phase sampling DRFM comparator technology. The phase sampling DRFM digitizes the phase of the I,Q components with the sampling period (time between phase samples) corresponding to the range resolution of the DRFM. The DRFM phase data is fed serially into the tapped delay processor with each delay corresponding to the range resolution of the image synthesizer. The phase data at each tap is processed in a pipelined range bin signal processor in order to generate the selected scattering mechanism. As previously discussed, this is done by continuously rotating the phase $n\Delta\phi = n\omega_d PRI$, translating the phase into a complex signal I,Q that is amplitude modulated using A_i. When the complex I,Q data exits each tap it is summed with available data from all the other tap processors *each clock cycle*. The digital sum at each clock cycle is then converted to an analog signal for up conversion onto the carrier for retransmission.

In order to show the equivalence of both architecture variations, the details of the original architecture for the in-phase processing is shown in Figure 4.2 where E is the image extent, $\Delta\phi_i$ is the phase increment value for the i^{th} tap processor, and A_i is the amplitude modulation. The input phase is $\phi(n)$ and the output is

$$I(n) = \sum_{i=0}^{E} A_i \cos(\phi(n-i) + \Delta\phi_i) \tag{4.1}$$

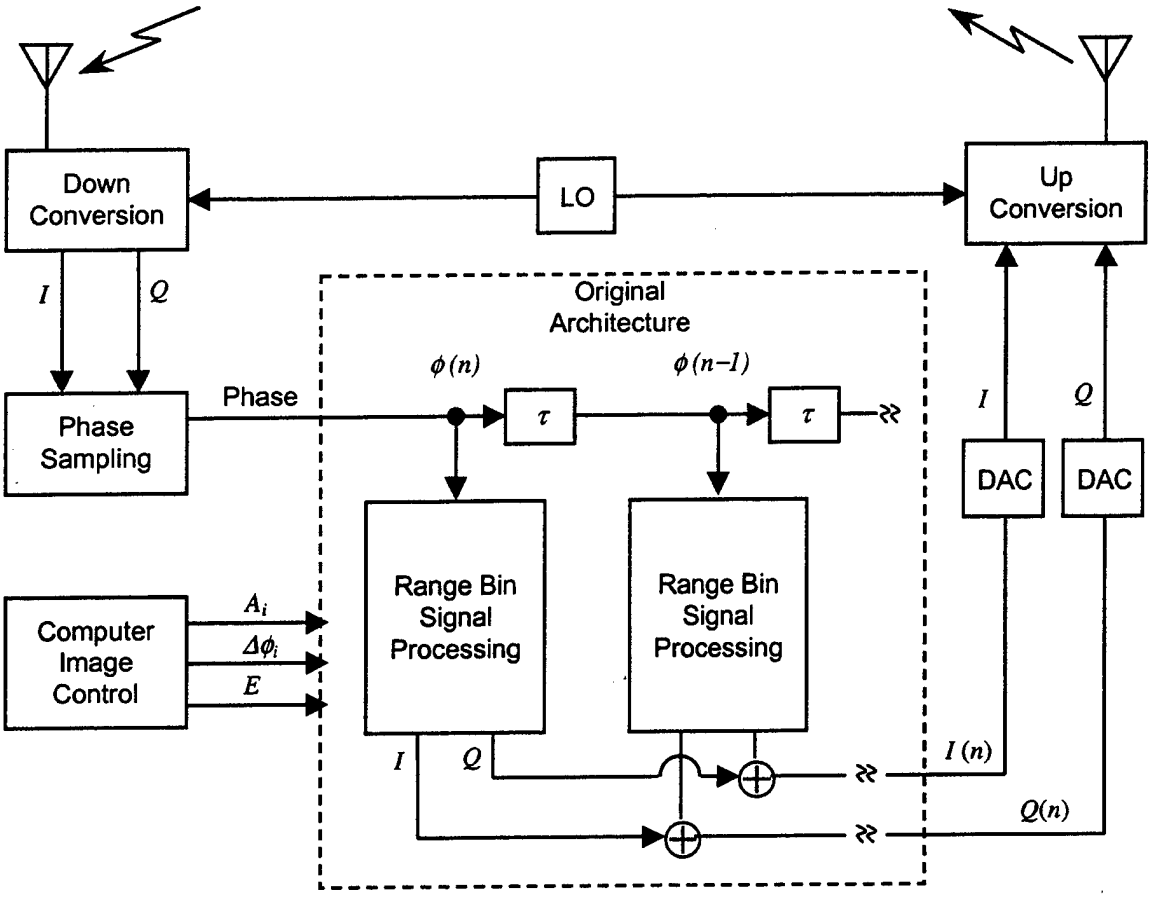


Figure 4.1: Block diagram of the original DIS architecture.

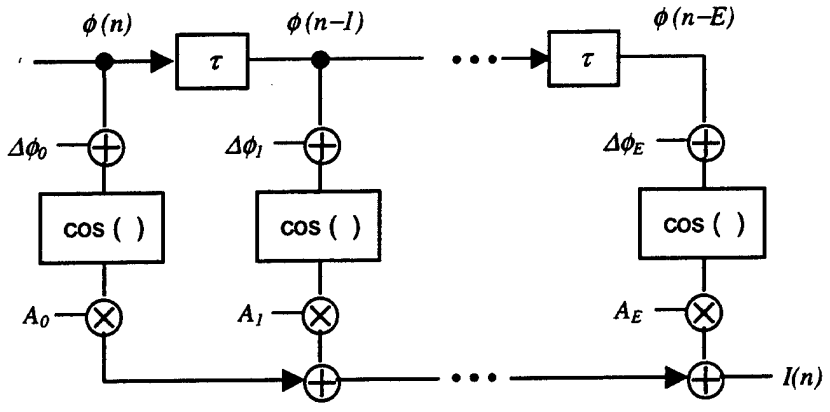


Figure 4.2: Original DIS architecture for in phase processing.

The "modified architecture" was developed while investigating a move from field programmable gate array (FPGA) technology (Altera's Max+Plus II) to an application specific integrated circuit (ASIC). A block diagram of the modified architecture is illustrated in Figure 4.3.

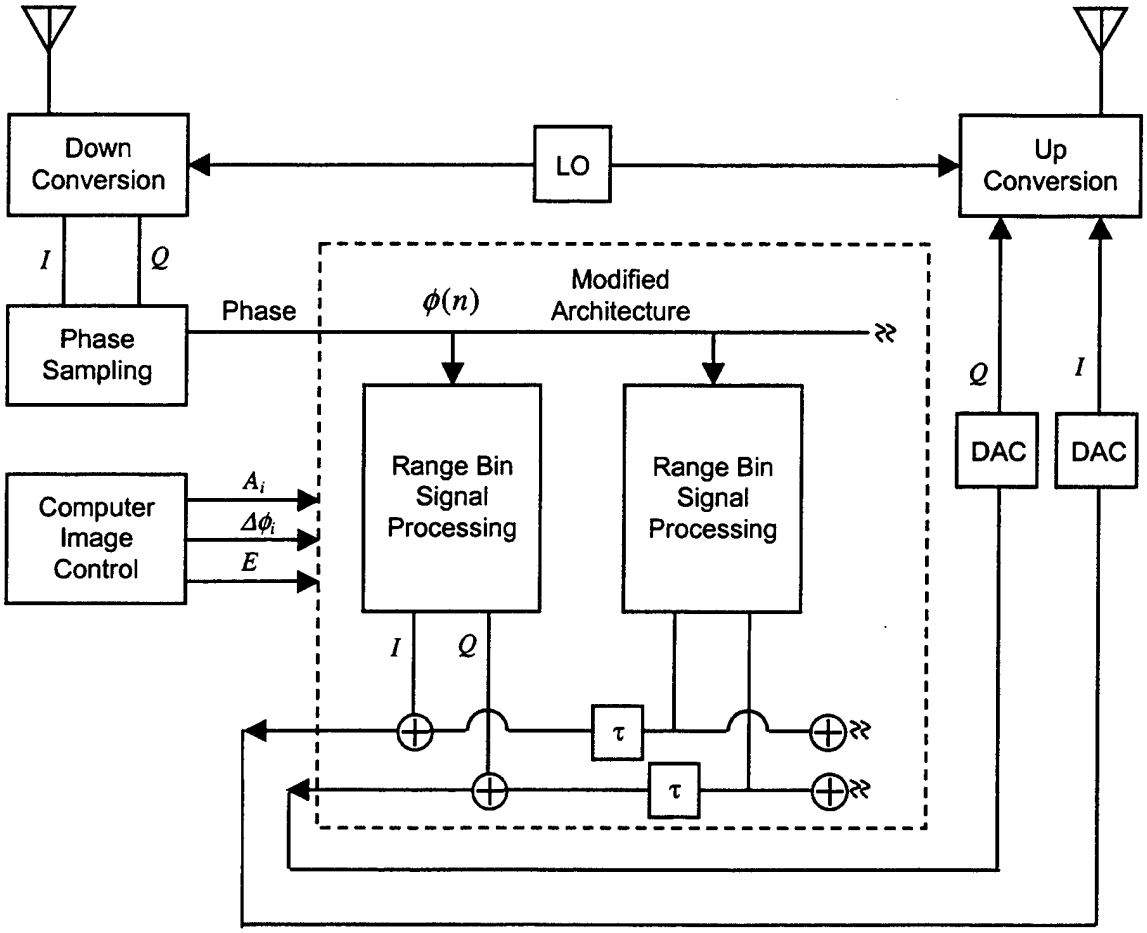


Figure 4.3: Block diagram of the modified DIS architecture.

The modified algorithm enables loading all tap processors synchronously with the DRFM phase data. The DRFM phase data is processed in parallel in all tap processors in a pipelined fashion. The results from the taps are then added together by partial sums (serial summation) from one tap to another. The major difference between the original architecture and the modified architecture is that the time delay processor is embedded within the summation at the output. For both of the approaches described above it is essential that the individual taps be sequentially enabled during

the start-up or initial strobing of the phase data from DRFM into the tapped delay line. The taps must also be sequentially disabled during shutdown as the phase data leaves the DIS. This avoids the problem of erroneous data from entering into the summation during start-up and shutdown. More details concerning the change of technology is addressed in Chapter 7. The details for the modified DIS are shown in Figure 4.4

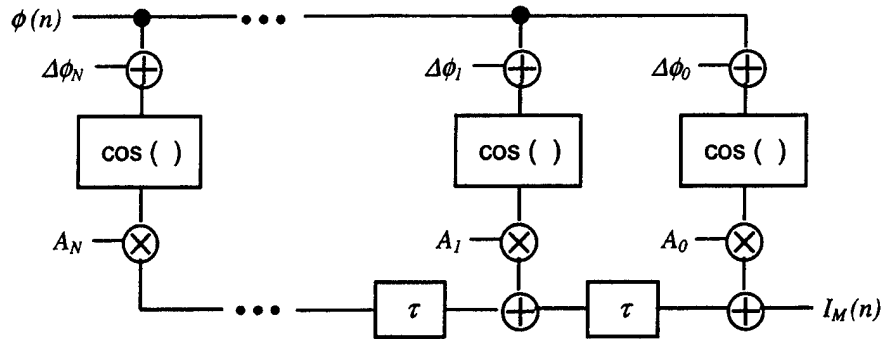


Figure 4.4: Modified DIS architecture for in phase processing.

$$I_M(n) = A_0 \cos(\phi(n) + \Delta\phi_0) + D^1[A_1 \cos(\phi(n) + \Delta\phi_1)] + \cdots + D^N[A_N \cos(\phi(n) + \Delta\phi_N)] \quad (4.3)$$

where D is a delay operator. Rewriting

$$I_M(n) = A_0 \cos(\phi(n) + \Delta\phi_0) + A_1 \cos(\phi(n-1) + \Delta\phi_1) + \cdots + A_N \cos(\phi(n-N) + \Delta\phi_N) \quad (4.4)$$

or

$$I_M(n) = \sum_{i=0}^{N} A_i \cos(\phi(n-i) + \Delta\phi_i) \quad (4.4)$$

which is exactly (4.1).

B. Simulation Overview

To evaluate the performance of the architecture and to compare the results of the hardware implementation, a MATLAB simulation was constructed of both the DIS and an ISAR as shown in Figure 4.5. Some of the essential features of an ISAR are simulated including the

wideband chirp pulse waveform that is intercepted by the DIS. The DRFM/DIS is also simulated. The complex outputs from the DRFM/DIS are presented to the ISAR signal processing for image generation. MATLAB has also been used in several intermediate steps to be able to compare simulation results with actual and simulated hardware design results.

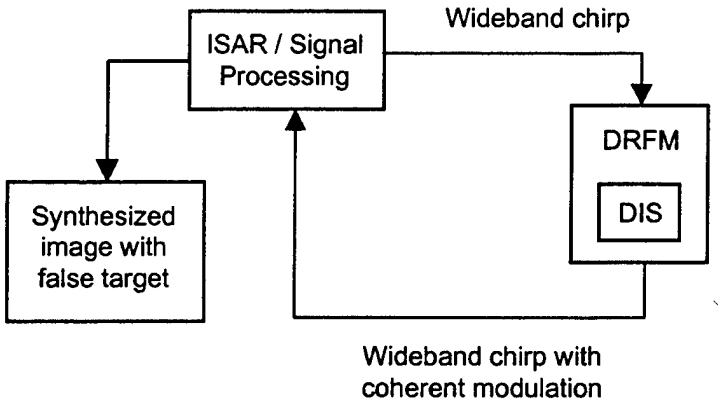


Figure 4.5: ISAR-DIS simulation configuration.

MATLAB is a product from the MathWorks, Inc. and it is an integrated technical computing environment that combines numeric computation, advanced graphics and visualization, and a high-level programming language [5].

MATLAB includes several useful functions for:

- Data analysis and visualization
- Numeric and symbolic computation
- Engineering and scientific graphics
- Modeling, simulation, and prototyping
- Programming, application development, and graphical user interface (GUI) design

MATLAB can be used in a variety of application areas including signal and image processing, control system design, financial engineering, and medical research. It features a family of application-specific toolboxes, containing comprehensive collections of functions for solving particular classes of problems in areas such as signal processing, image processing,

control system design, neural networks, and more. The current version of MATLAB used in this project is V.5.3.

In FY98, Siew-Yam Yeo developed the original set of codes during his thesis work at the Naval Postgraduate School [6]. This set of codes has been modified to better serve the purpose of further development in the project. For example, the "original" code has been modified to deal with more then three taped delay lines. This set of codes all end with a "*...v1.m*" extension. Parallel to the development of the ASIC hardware design (modified DIS architecture), simulations were developed to emulate the new design. The new codes are used to verify that the new modified architecture is giving the correct results. This set of codes all end with a "*...v2.m*" extension.

C. Simulation Details

Using numerous comments, the different steps of the simulation are easily identified within the set of simulation codes (m-files). A description of the steps, together with some intermediate results are given below in order to visualize the development process. The flowchart shown in Figure 4.6 together with Table 4.1 summarize the different MATLAB files used during the simulation. Important text files used in the intermediate steps are also listed.

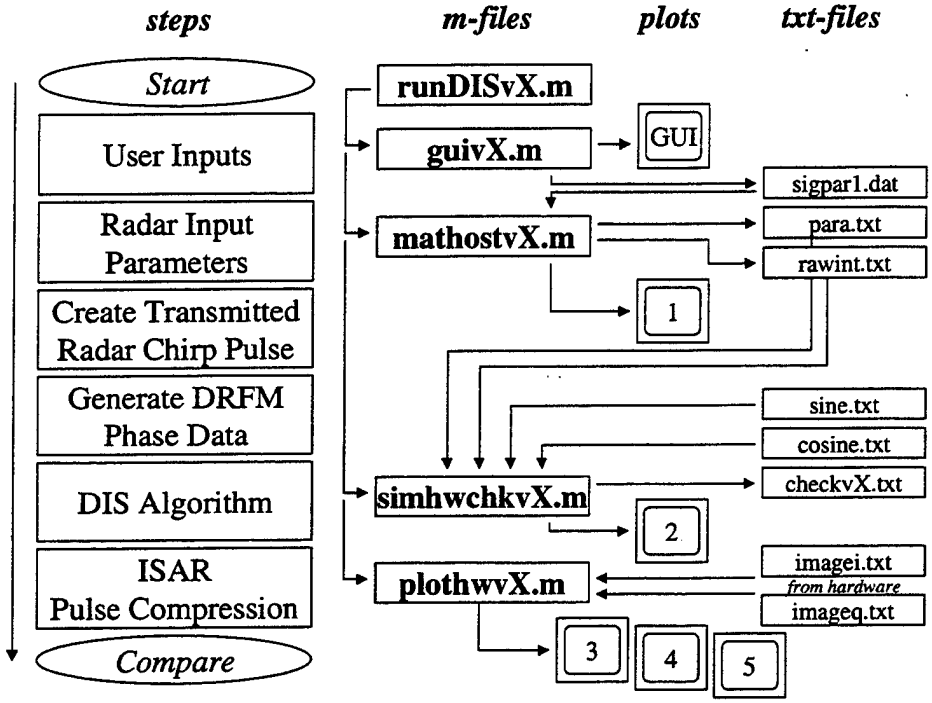


Figure 4.6: MATLAB simulation flowchart.

m-files	*txt-files*	*Remarks*
runDISvX.m		– To execute the simulation
guivX.m		– To get user inputs of the false target to be generated
	sigpar1.dat	– Signal parameters of the false target to be generated
mathostvX.m		– Simulates the ISAR transmitted pulse – Simulates the DRFM at the DIS location
	para.txt	– Number of range bins of the ISAR – Number radar pulses to be processed (integrated) – Target extent – Amplitude settings for each cell – Phase values representing an increasing/decreasing Doppler shift
	rawint.txt	– DRFM phase data samples
simhwchkvX.m		– Simulates the DIS algorithm
	cosine.txt	– Cosine look-up table, 32 values for one period
	sine.txt	– Sine look-up table, 32 values for one period
dec2two.m		– MATLAB function that converts decimal number to 2-complement binary representation
two2dec.m		– MATLAB function that converts 2-complement binary representation to decimal number
	checkvX.txt	– Intermediate results through the DIS algorithm
	imagei.txt	– Hardware/hardware simulation results (I-channel)
	imageq.txt	– Hardware/hardware simulation results (Q-channel)
plothwvX.m		– Pulse compresses the radar return of the false target generated by the DIS hardware – Plots the final results for comparison

Table 4.1: Files used during the MATLAB Simulation

The *m-files* mentioned in the table above together with the *cosine.txt* and the *sine.txt* files are attached in Appendix A.

User input: To run the simulation the user executes the *runDISv1.m* or the *runDISv2.m* file depending whether the *original* or the *modified* architecture is desired (afterwards the files are referred to as "*...vX.m*"). The *runDISvX* program is a script file to execute other script files in a pre-defined order. The user is presented with a graphical user interface (GUI) of a Range/Doppler map – the Range-Doppler-Amplitude Map Entry Program *guivX.m* is shown in Figure 4.7 (*runDISvX.m* executes *guivX.m*).

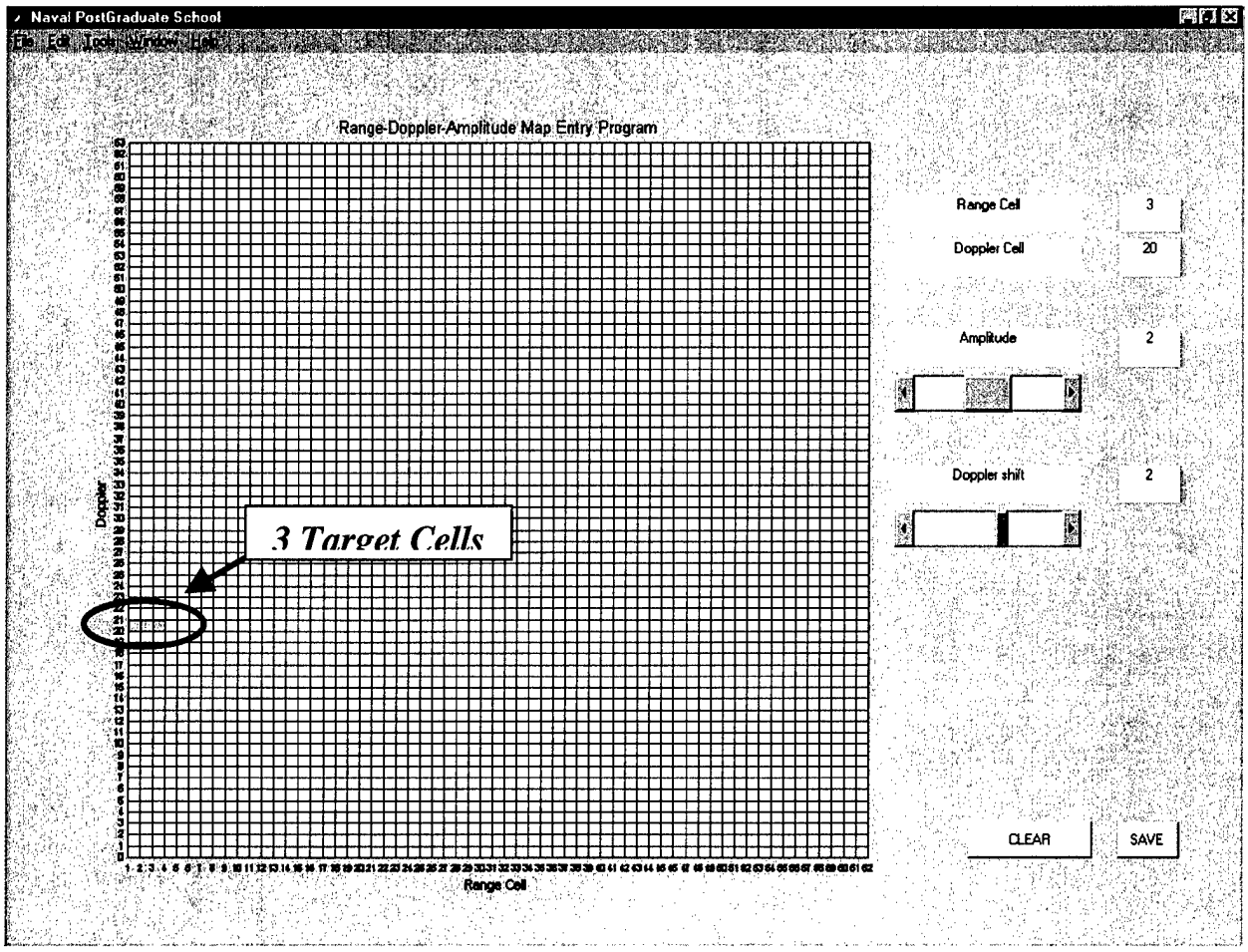


Figure 4.7: The range-Doppler-amplitude map entry program.

In this example the user has specified the following data to generate the false target using the DIS shown in Table 4.2.

Target Cell	*Range Cell*	*Doppler Cell*	*Amplitude*	*Doppler Shift*	*Remark*
1	1	20	2	0	Tap 0 – 1st Tap
2	2	20	2	1	Tap 1 – 2nd Tap
3	3	20	2	2	Tap 2 – 3rd Tap

Table 4.2: User Specified Inputs of the False Target

The values, called signal parameters of the false target, are written to an intermediate file that is called *sigparl.dat*. Examining the *sigparl.dat* file for this case will give the values shown in

Table 4.3. The file only holds the numerical values. The header of the table has been applied later to explain what the different values relate to.

Range Cell	*Doppler Cell*	*Amplitude*	*Doppler Shift*
1.0000000e+000	2.0000000e+001	2.0000000e+000	0.0000000e+000
2.0000000e+000	2.0000000e+001	2.0000000e+000	1.0000000e+000
3.0000000e+000	2.0000000e+001	2.0000000e+000	2.0000000e+000

Table 4.3: Contents of the file *sigpar1.dat.*

Defining the Radar Parameters: The next file to be executed by the *runDISvX.m* file is *mathostvX.m*. The *mathostvX.m* file represents both the ISAR while generating the transmitted chirp pulse and the DRFM on the platform where the DIS is located. The radar specific parameters of the ISAR are coded into this program. In this case the radar parameters used is shown in Table 4.4.

ISAR Theoretical Parameter	*Value*	*MATLAB Equivalent Variable*
– Uncompressed pulse width, τ	500 ns	*pw*
– Compressed pulse width, τ_c	8 ns	*pwc = 1/(1.25 x bw)*
– Pulse repetition frequency, *PRF*	2 kHz	*prf*
– Pulse repetition interval, *PRI*	500 μs	*pri*
– Bandwidth of the chirp pulse, *BW*	125 MHz	*bw* = 100 MHz
– Pulse compression rate, *K*	2.5 x 10^{14}	*mu* = $2\pi(bw/pw)$ = 1.256 x 10^{15}
– Sampling frequency, f_s	125 MHz	*fs = 1.25 x bw* = 125 MHz
– Sampling time step, t_s	8 ns	*Ts = 1/fs* = 8 ns

Table 4.4: Defined Radar Parameters (file *mathostvX.m*).

Creation of the Intercepted Radar Signal: The signal parameters specified by using the GUI used to create the baseband complex signal represented by

$$S_b(t) = \text{rect}\left(\frac{t}{T}\right) e^{j2\pi\left(f_d PRI + Kt^2/2\right)} \tag{4.5}$$

where f_d is the Doppler frequency of the DIS platform intercepting the chirp signal. Note that this expression is similar to (2.1) where the parameter K is the chirp slope rate and T is the pulsewidth. The Doppler frequency f_d has to be taken into consideration when building the received chirp waveform in the DIS simulation. An approximation is used that assumes a constant phase change due to Doppler within a chirp pulse. This assumption is valid since the Doppler shift is only tens of hertz compared to the MHz chirp bandwidth. The wideband intercepted signal is then phase sampled and the phase is quantized into 5-bits or 32 different values, representing a phase between 0 and 2π radians. The values used are 0 to 31 as a decimal representation of a 5-bit binary word ($2^5 = 32$). The DRFM phase data is written to a text file (rawint.txt) that is read by *simhwchkvX.m*. An example of the DRFM phase data matrix contained in the *rawint.txt* file is shown in Table 4.5. The file only holds the numerical values. The rows of the matrix represent radar pulses. The columns represent DRFM phase data samples from a specific radar pulse at specific sampling times. The variable names used in MATLAB are also shown.

Radar Pulse *(batchCnt)*	***DRFM Phase Data*** *(intraPulseCnt)*													
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	.	.	.	*62*
1	0	0	0	0	0	0	5	5	10	15	.	.	.	10
2	15	15	15	15	15	15	15	20	20	25	.	.	.	25
3	25	25	25	25	25	25	31	31	4	9	.	.	.	4
4	10	10	10	10	10	10	10	15	15	20	.	.	.	20
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
64	25	25	25	31	31	31	31	4	4	0				9

Table 4.5: Contents of the file *rawint.txt*.

The impulse response waveform used in the ISAR range compression algorithm is also computed when executing this file. The amplitude and Doppler frequency *shift* values for each range Doppler cell are also obtained from the GUI and represent the gain and phase rotation values required for the DIS.

A number of different values are written to another text file (*para.txt*). The values are used for simulating the DIS both in MATLAB and in the hardware design. The file only holds the numerical values. These values represent the following information (also exemplified in Table 4.6).

- number of range bins of the ISAR
- number radar pulses the ISAR is using for processing (integrating) received radar return signals
- target extent (number of target cells/taps used)
- amplitude settings for each cell translated into a gain value of 1, 2, 4 or 8
- set of phase values representing an increasing/decreasing Doppler shift due to the motion of the target cell relative to the ISAR

Value	***Variable***	***Comment***
62	*nRangeCell*	Number of Range Cells (Range Bins of the ISAR)
64	*nDopplerCell*	Number of Doppler Cells (Doppler Bins of the ISAR)
3	*targetExtent*	Target Extent
2	*gain(1)*	Gain modulation coefficient, target cell 1
2	*gain(2)*	Gain modulation coefficient, target cell 2
2	*gain(3)*	Gain modulation coefficient, target cell 3
0	*phi(1,batchCnt)*	Doppler modulation coefficient, target cell 1, 1st radar pulse
0	*phi(2,batchCnt)*	Doppler modulation coefficient, target cell 2, 1st radar pulse
0	*phi(3,batchCnt)*	Doppler modulation coefficient, target cell 3, 1st radar pulse
0	.	Doppler modulation coefficient, target cell 1, 2nd radar pulse
1	.	Doppler modulation coefficient, target cell 2, 2nd radar pulse
1	.	Doppler modulation coefficient, target cell 3, 2nd radar pulse
0	.	Doppler modulation coefficient, target cell 1, 3rd radar pulse
1	.	Doppler modulation coefficient, target cell 2, 3rd radar pulse
2	.	Doppler modulation coefficient, target cell 3, 3rd radar pulse
.	.	.
.	.	.
.	.	.
0	*phi(1,batchCnt)*	Doppler modulation coefficient for target cell 1, 64th radar pulse
31	*phi(2,batchCnt)*	Doppler modulation coefficient for target cell 2, 64th radar pulse
63	*phi(3,batchCnt)*	Doppler modulation coefficient for target cell 3, 64th radar pulse

Table 4.6: Contents of the file *para.txt*.

To visualize the effect of the amplitude and the Doppler frequency shift values shown in Figure 4.7, the range-Doppler image from the ISAR signal-processing simulation is plotted in Figure 4.8.

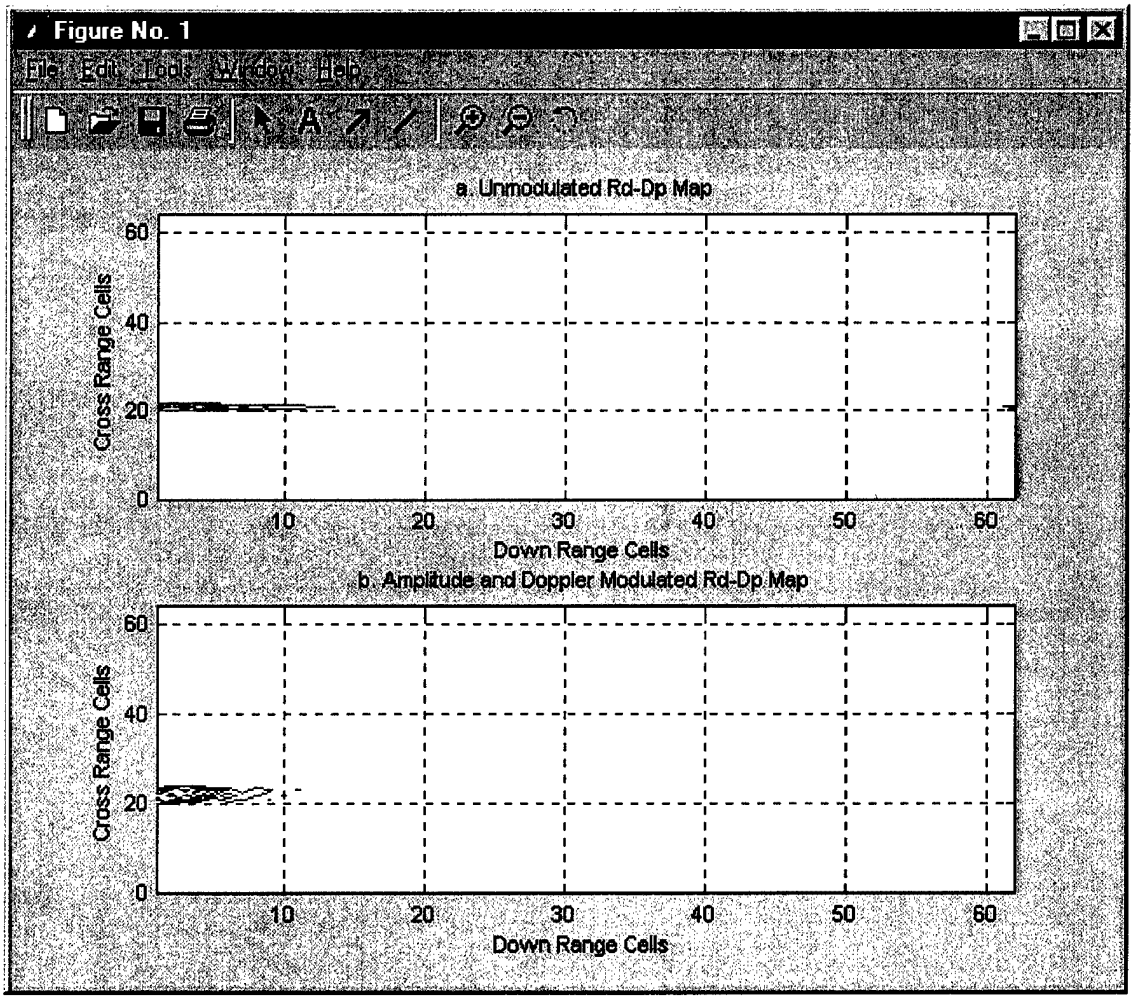


Figure 4.8: ISAR range-Doppler image with (a) no amplitude or Doppler frequency shift and (b) amplitude and Doppler frequency shift as shown in Table 4.2.

Figura 4.8 (a) represents the ISAR range-Doppler image but contains no amplitude or Doppler frequency shift. Figure 4.8 (b) shows the ISAR range-Doppler image with amplitude and Doppler frequency shift as shown in Table 4.2.

Simulation of the DIS (Original and Modified Architecture): To simulate the DIS algorithm the *runDISvX* program executes the *simhwchkvX.m* file, which starts by reading in the values from *para.txt*. The number of Doppler cells within the range-Doppler map is used as an index for an outer for-loop in the program for processing phase data from one radar pulse to the next. The

number of range bins within the range-Doppler map is used as an index for an inner (nested) for-loop and represents the number of clock pulses it takes to process the DRFM phase data from one radar pulse to the next. The target extent represents the number of taps in the tap delay line. A target cell is also referred to as a tap in the DIS algorithm. The number of target cells specified in the GUI is therefore equivalent to the number of taps used to create a false target. The gain value selected for each tap along with the corresponding Doppler frequency shift are recorded and relate to the synthesized motion of each target cell.

Next the DRFM phase data from the *rawint.txt* file is read. The program also loads data from *cosine.txt* and *sine.txt*. These files hold data used as the look-up table (LUT) and contain one period of a cosine waveform and a sine waveform (32 values) as shown in Figure 4.9. Recall that the LUT translates the input phase (from the phase accumulator) into a complex signal. Using the for-loops the DIS algorithm modulates the phase data to compute the signal that represents the return signal corresponding to the desired false target. The original and the modified architecture calculate the modulation and perform the computation in different ways as described earlier.

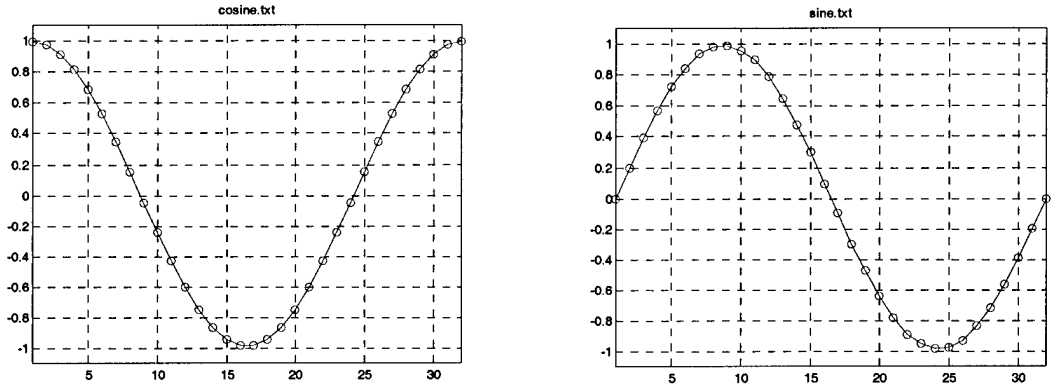


Figure 4.9: Cosine and sine look-up table (LUT).

In the original DIS architecture the DRFM phase data propagates serially from tap to tap during one clock pulse time delay. The phase data at each tap is then modulated and the results from all taps are summed together to form the output. In the modified DIS architecture, the DRFM phase data is presented to all the taps synchronously. The phase data in this case, is processed in parallel in all taps. The delay is implemented during summation of the results from

each tap. The individual taps are enabled during the start-up and disabled during shut-down according to the reasons described earlier.

In the MATLAB simulation several sets of DRFM phase data representing samples from a number of radar pulses are processed directly one after another. In an actual implementation the set of DRFM phase data will of course be separated in time by one PRI.

Range and Azimuth Compression: At the receiver side of the ISAR, as part of the signal processing, the radar return signals containing the generated false target are compressed both in range and azimuth. First range compression is done. Range compression is based on correlating the received signal with a pre-stored reference waveform and neglecting the complex amplitude (see (2.6) and Figure 2.4)

$$h(n) = e^{j\pi K(n\Delta t)^2} \tag{4.6}$$

The FFT is performed on the received signal. The resulting spectrum is multiplied by the complex conjugate of the FFT of the reference waveform (4.6) created in the *mathostvX.m* file. An inverse FFT (IFFT) is then performed to obtain the range bin profiles for each PRI.

For the azimuth compression for a single range bin the complex range samples are taken from 2^n pulses and integrated into an FFT. The magnitude of the FFT output is the Doppler profile for that particular range bin.

The received signal after compression can be visualized as a contour plot as shown in Figure 4.10 and is referred to in the second sub-plot below as the MATLAB Simulation plot (Amplitude and Doppler Modulated Range-Doppler Map).

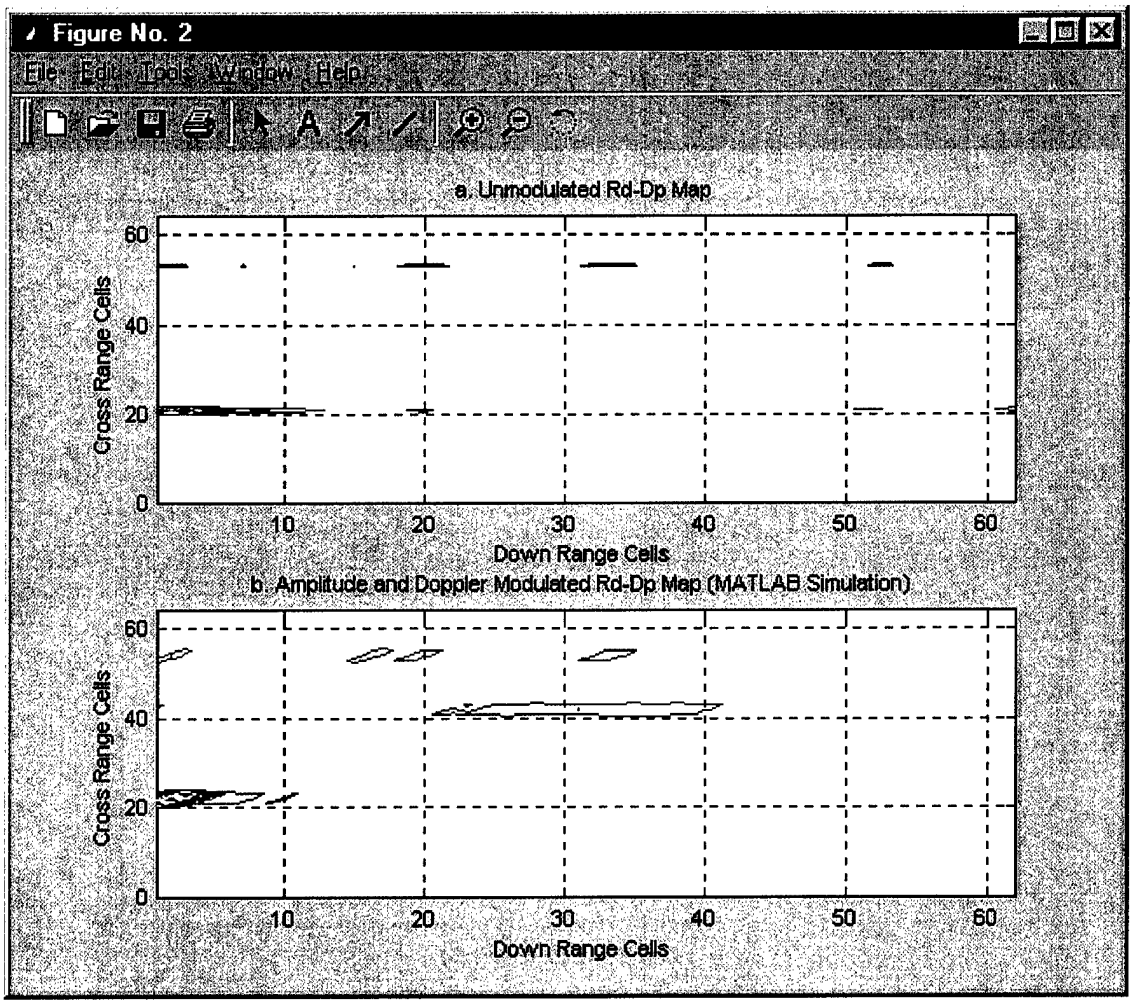


Figure 4.10: ISAR range-Doppler images showing (a) the unmodulated DIS output and (b) the modulated DIS output (MATLAB Simulation).

Plot and Compare Results: The last file to be executed by the *runDISvX* program is the *plothwvX.m*. This file obtains the I- and Q-values of the hardware simulation from the *imagei.txt* and the *imageq.txt* file (written by Altera/Visual Basic FPGA hardware program). Range and azimuth pulse compression is performed using the same procedure as described for the MATLAB simulation results. The results are plotted for comparison. The DIS simulation results are shown in the first sub-plot of Figure 4.11. In the second sub-plot the hardware or hardware simulation

result are shown when data is available. A full comparison is shown in the following Chapters when the different hardware implementation techniques are described.

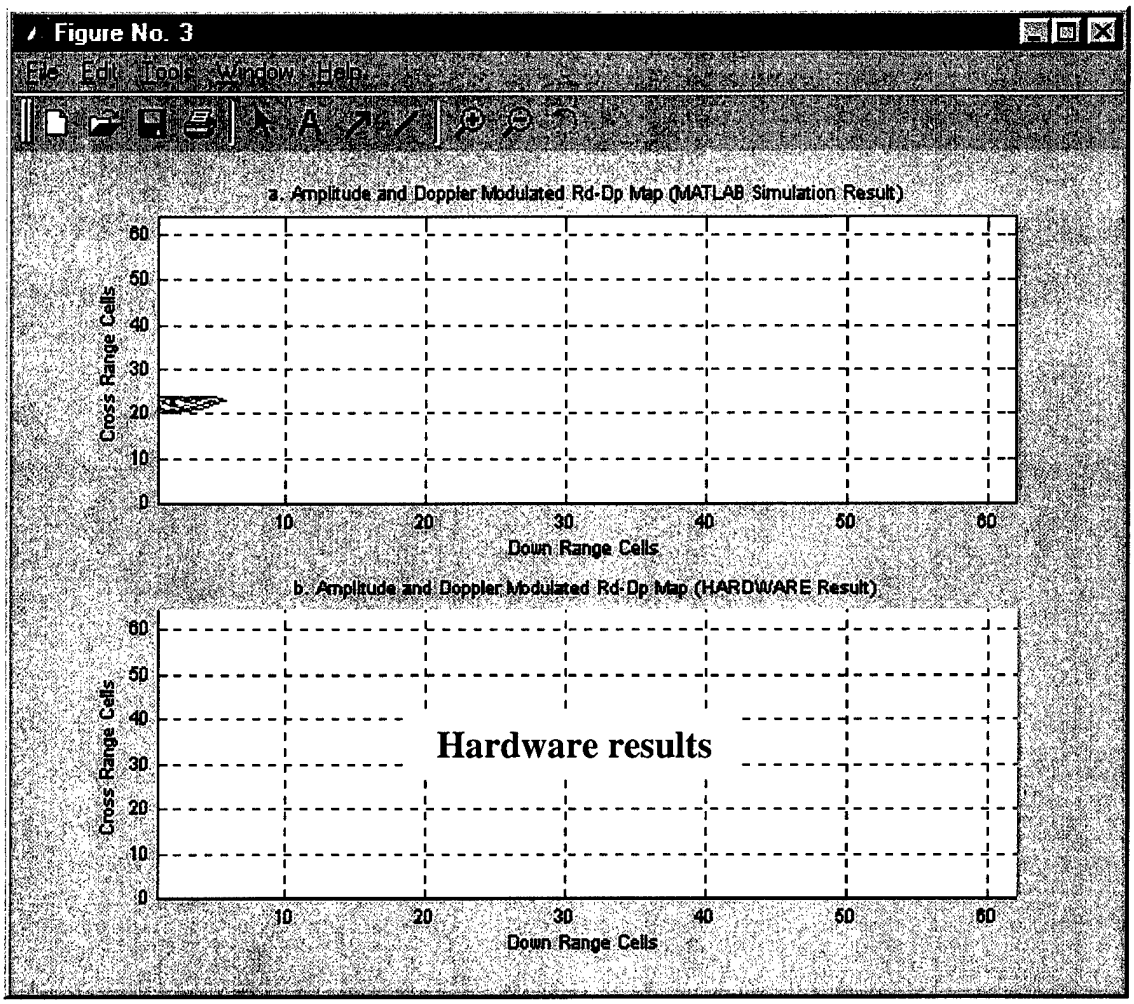


Figure 4.11: MATLAB DIS simulation versus hardware results.

To better visualize the image created by the DIS (the generated false target seen by the ISAR), MATLAB uses the same data as before to construct a 3-D mesh surface plot, as shown in Figure 4.12. The first sub-plot shows the result from the MATLAB DIS simulation. The second subplot shows the hardware (or hardware simulation) result. Finally, the third sub-plot shows the difference between the MATLAB simulation and the hardware results.

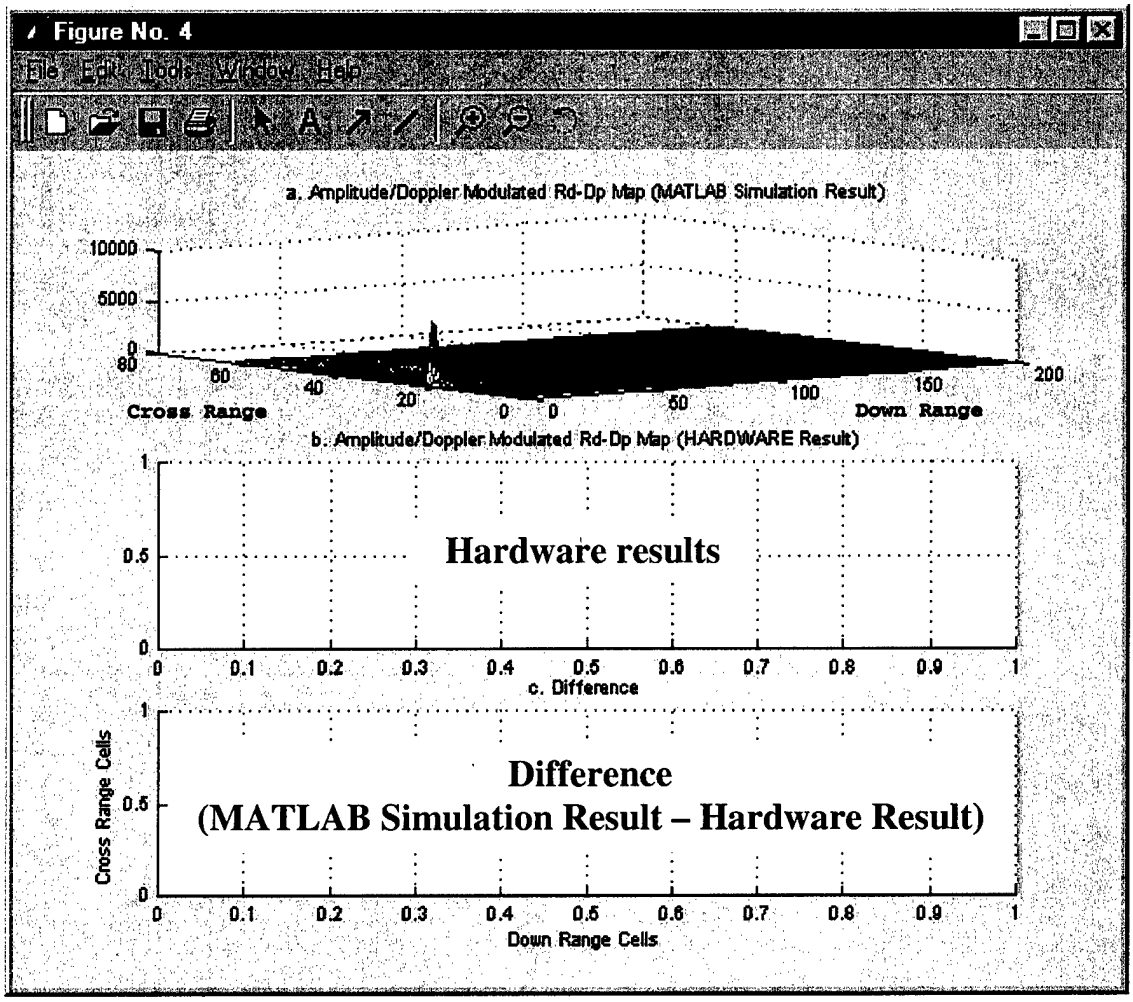


Figure 4.12: MATLAB simulation result versus hardware result, and the difference.

To better study the results from the DIS simulation, the ISAR image of the false target is exposed, as shown in Figure 4.13. The user defined target cells, after DIS modulation and ISAR signal processing (range and azimuth compression) stand out clearly from the background in the plot.

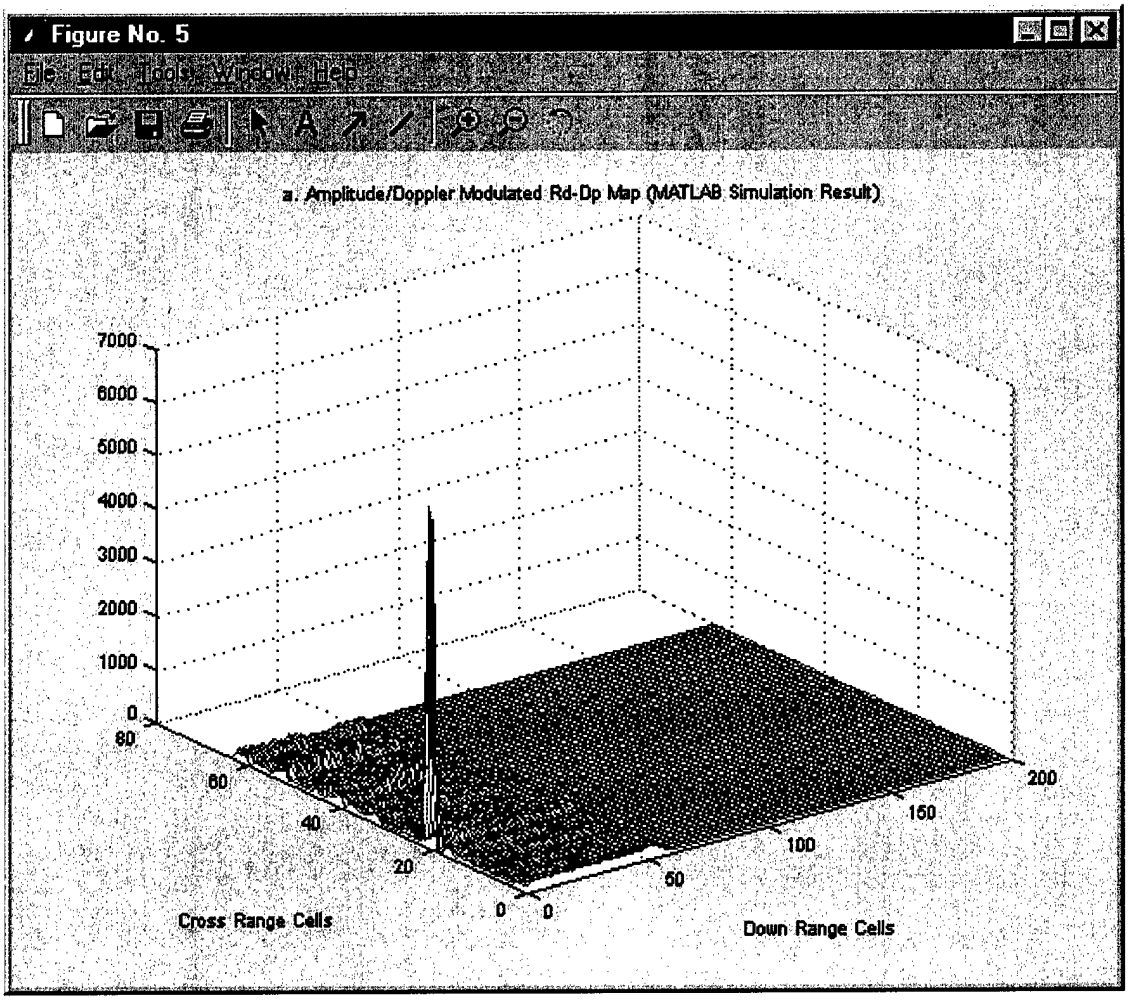


Figure 4.13: MATLAB simulation result (3-D mesh surface plot).

Original and Modified DIS Comparison: To ensure that both the original and modified algorithms produce the same result, a series of comparisons for different test cases were conducted. The example below shows the ISAR output when using the different algorithms. It also shows the ability to modulate the extent of the false target using a large number of taps. In the test case below 32 taps are used. Figure 4.14 shows the input target entry. Table 4.7 shows the amplitude and Doppler offset values selected for the 32 range bin false target to be synthesized.

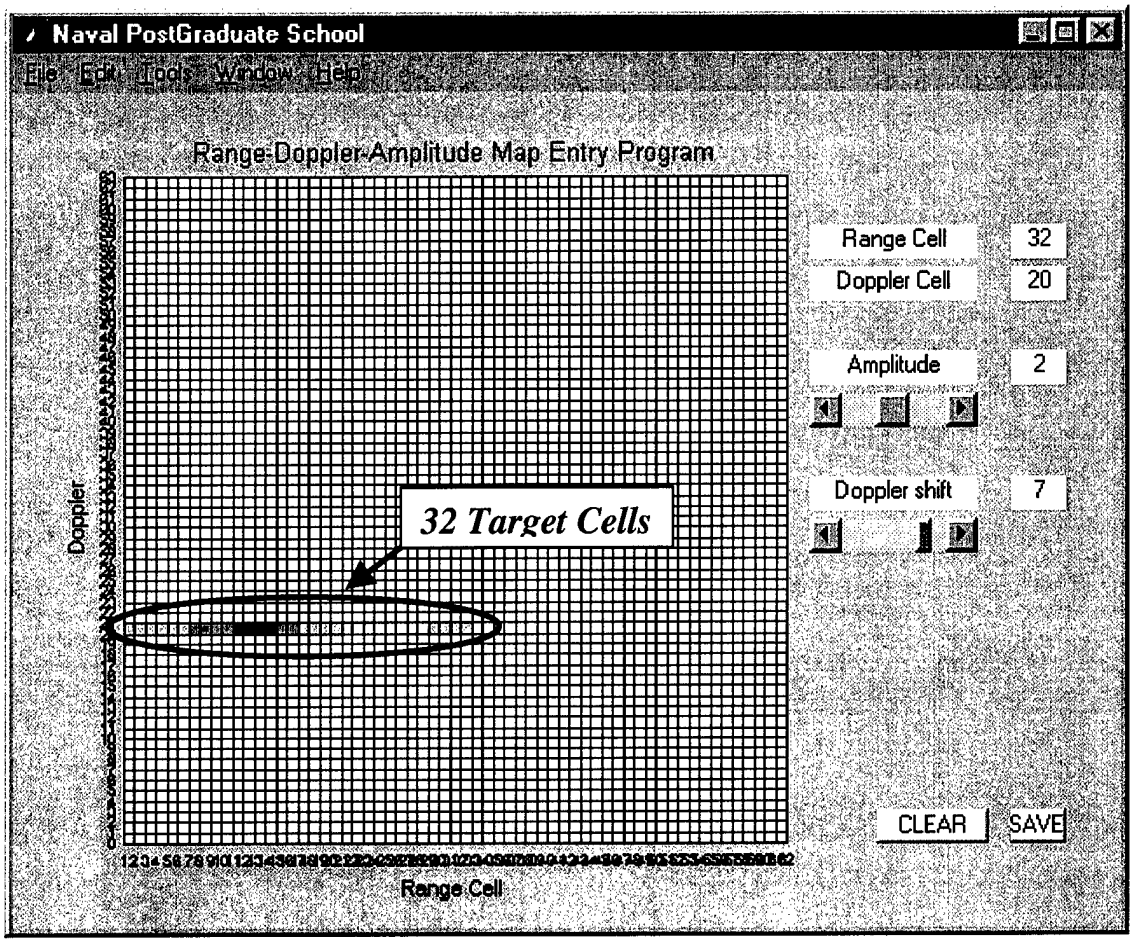


Figure 4.14: The range-Doppler-amplitude map entry program.

	1-2	3-4	5-6	7-8	9-10	11-12	13-14	15-16	17-18	19-20	21-22	23-24	25-26	27-28	29-30	31-32
Amp	2	2	2	3	3	4	4	3	2	2	1	1	1	1	2	2
Dp	-8	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7

Table 4.7: Amplitude and Doppler Offsets Selected for 32 Range Bin False Target

As observed in Figures 4.15 and 4.16, the two different algorithms perform the same result.

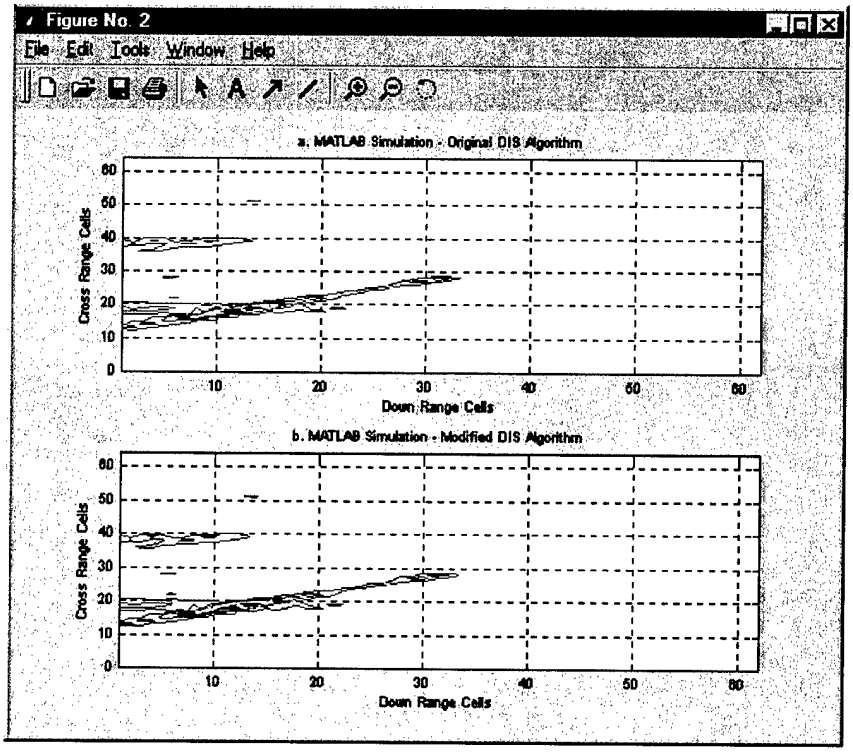


Figure 4.15: Original versus modified algorithm DIS simulation results.

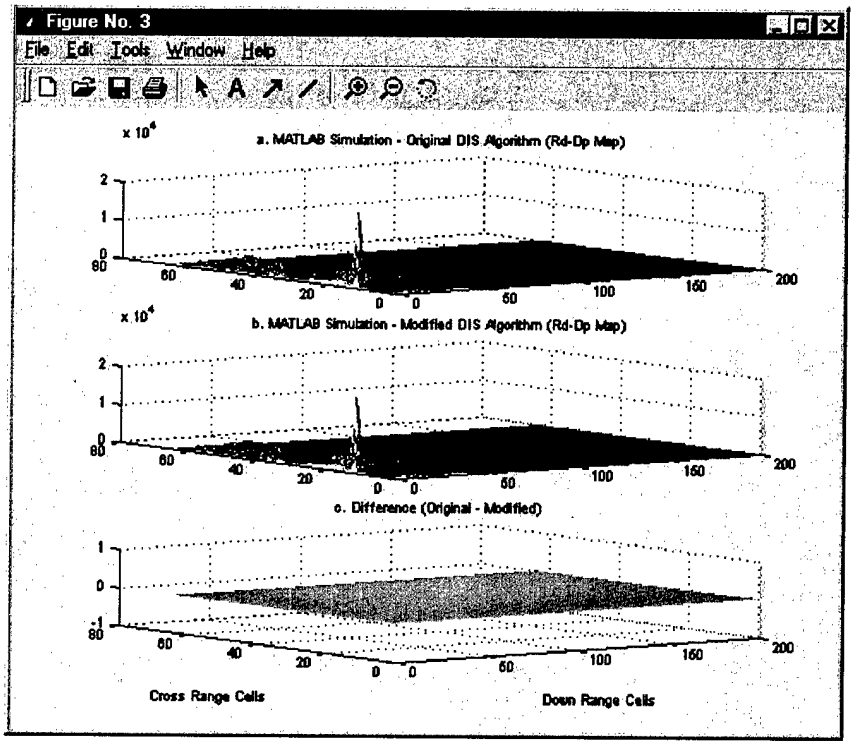


Figure 4.16: Original versus modified algorithm DIS simulation results, and the difference.

5. DIS USING FIELD PROGRAMMABLE GATE ARRAYS

A. Introduction

This chapter discusses the hardware implementation of the DIS by using FPGA technology. The hardware design is captured using the Altera Multiple Array Matrix Programmable Logic User System or MAX+PLUS II software version 9.21 (the project was started in 1998 using version 8.3). MAX+PLUS II is the design environment for Altera programmable logic devices (PLD). A brief description of the MAX+PLUS II software is given below followed by a short introduction to Field Programmable Logic Devices (FPLDs) [7]. In particular, FPGAs, specifically the Altera 10K50 family are described. Later sections of this chapter describe each of the modules of the DIS hardware design, starting from the top-level-hierarchy and progressing down. The final section addresses the FPGA results and the comparison to MATLAB simulations.

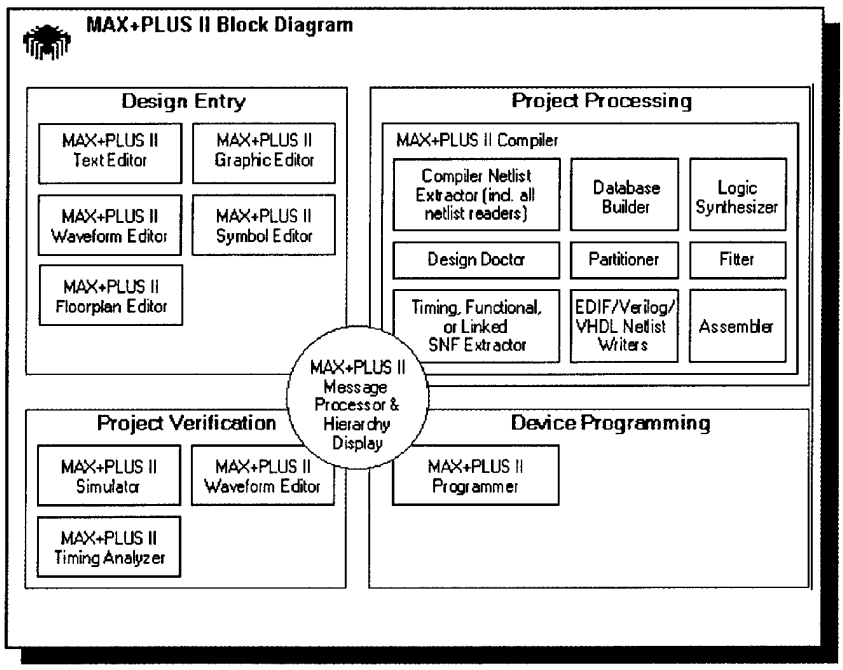


Figure 5.1: Altera MAX+PLUS II environment from [8].

B. The Altera MAX+PLUS II Environment

The MAX+PLUS II software provides a multi-platform, architecture-independent design environment that easily adapts to specific design needs. The MAX+PLUS II development software is a fully integrated programmable logic design environment. This tool supports all Altera programmable device families and works in both PC and UNIX environments. The MAX+PLUS II allows seamless integration with industry-standard design entry, synthesis, and verification tools. Figure 5.1 shows a block diagram of the Altera MAX+PLUS II environment. MAX+PLUS II both reads and writes:

- Altera Hardware Description Language (AHDL) files and standard EDIF netlist files
- Verilog HDL files
- VHDL files
- OrCAD schematic files

In addition, MAX+PLUS II reads Xilinx netlist files and writes Standard Delay Format (SDF) files for interface to other industry-standard CAE software. The MAX+PLUS II message processor handles the different features like design entry, project processing, project verification and device programming. An overview of the MAX+PLUS II compiler interface is shown in Figure 5.2. The hierarchy display is a convenient way to switch between the different parts of the program and shows a hierarchy tree with branches, that represents the sub designs.

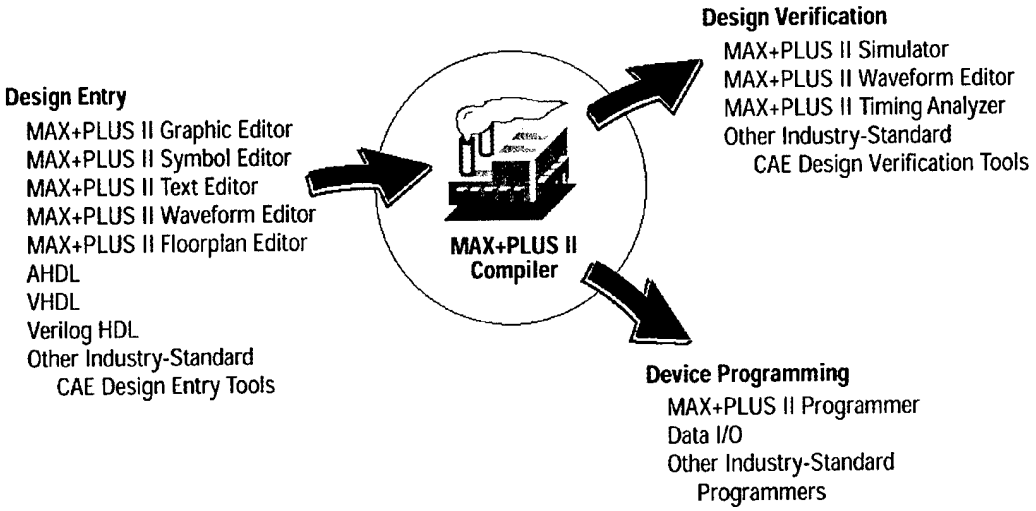


Figure 5.2: MAX+PLUS II design environment from [9].

The complete MAX+PLUS II system includes 11 fully integrated applications that take the designer through every step of creating a design. A logic design, including all sub-designs, is called a "project" in MAX+PLUS II. The main applications are summarized in Table 5.1.

Application	*Function*
Hierarchy display	For displaying the current hierarchy of files as a hierarchy tree with branches, that represents sub designs.
Graphic editor	For entering a schematic logic design. Altera provides primitives, megafunctions, and macrofunctions, which serve as basic circuit-building blocks.
Symbol editor	For adding existing symbol and creating new ones.
Text editor	For creating and editing text-based logic design files written in hardware description language (AHDL, VHDL, Verilog HDL).
Waveform editor	For entering test vectors and viewing simulation results.
Floor-plan editor	For assigning logic to physical device pins and logic cell resources in a graphic environment.
Compiler	For processing project, including checking for errors, synthesizing the logic, fitting the project into one or more Altera devices.
Simulator	For testing the logical operation and internal timing of logic circuits. The simulator supports functional simulations, timing simulations, and linked multi-device simulation.
Timing analyzer	For analyzing the performance of the logic circuits after it has been synthesized and optimized by the compiler.
Programmer	For programming, configuring, verifying, examining and testing Altera's devices.
Message processor	For displaying warning and information messages on the status of the project. It also locates the source of a message automatically in the original design files.

Table 5.1: MAX+PLUS II Suite of Applications and Functions from [8].

C. FPGA technology and the Altera 10K50

Different devices are available to capture the developed FPGA design file. The FLEX 10K50 chip (FLEX = flexible logic element matrix architecture) for example is a static random access memory (RAM) with typically 70,000 gates (logic & RAM). The Flex 10K50 device contains an embedded array and a logic array. The logic array performs the same function as a sea of gates in a gate array. It is used to implement general logic, such as counters, adders, state machines, and multiplexers. The embedded array is used to implement memory and specialized logic functions. Table 5.2 describes the features and benefits of using FPGAs and Table 5.3 the features of the FLEX 10K50. A picture of the FLEX 10K50 is shown in Figure 5.3.

Feature	*Benefit*
200 MHz and above system performance	Supports today's most demanding speed requirements
Density from 10,000 to over 1.5 million gates	Addresses 90% of all gate array design starts
Embedded array blocks	Efficient RAM, ROM, FIFO and other high-performance mega-functions
Multi-Volt I/O operation	Ideal for mixed-voltage systems
5.0V, 3.3V, 2.5V, and 1.8V device options	Supports multiple operating voltages
PCI compliance	Meets all specifications of the PCI local bus

Table 5.2: FLEX 10K Highlights from [9].

The Altera FLEX 10K devices are configured at system power-up with data stored in an Altera serial configuration EPROM device or provided by a system controller. A microprocessor interface that permits the microprocessor to configure the FLEX 10K devices serially, in parallel, synchronously or asynchronously supports the later [9].

Figure 5.3: Altera FLEX 10K50 device [9].

The features of the FLEX 10K50 device are as shown in Table 5.3:

Features	*FLEX 10K50*
System performance	115 MHz
Typical gates (logic & RAM)	50,000
Logic elements	2,880
Logic array blocks	360
Embedded array blocks	10
Total RAM bits	20,480
Flip-flops	3,184
Maximum user I/O pins	310

Table 5.3: Altera FLEX 10K50 Device Features from [9].

D. DIS Architecture using FPGAs

The Concept Demonstrator: A concept demonstrator of the DIS architecture has been developed in Field Programmable Gate Array (FPGA) technology. The concept demonstrator comprises three parts:

- Matlab simulations of the ISAR signal processing architecture (described in Chapter 4).
- Computer board containing hardware design using an Altera FPGA device (FLEX 10K50)
- A Visual Basic program (flextest.vbp) to access the Altera FPGA computer board and download the image-formation parameters and raw data, and upload from the board processed data. The data gathered from the board are stored in files that are in turn read by plothwvX.m for post-processing and display for comparison.

The DIS and its interface with the host computer is shown in Figure 5.4 as a block- and host-interface diagram. The host computer is an ordinary personal computer (PC). The DIS hardware is a FPGA (Altera 10K50 FPGA chip) mounted on a Naval Research Laboratory (NRL) custom designed PC I/O board. The various modules for the DIS are described below.

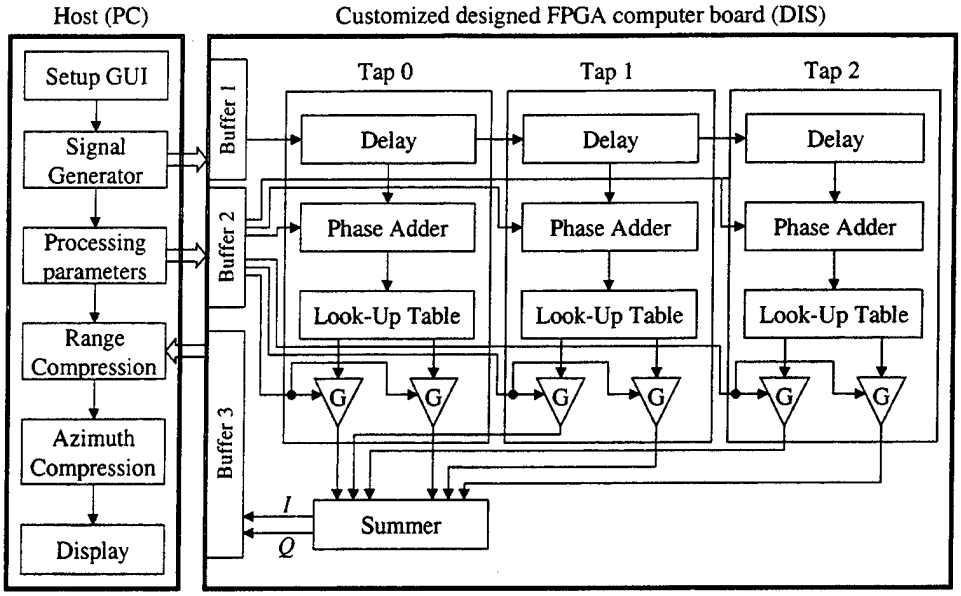


Figure 5.4: Block diagram and host-interface diagram of the DIS.

Host (PC)

Setup GUI: The setup as most of the blocks of the host refer back to the MATLAB code discussed in Chapter 4. In the GUI the user specifies the parameters for of the false target to be generated.

Signal Generator: The DRFM phase data samples are produced within this block and printed to a text file. This text file (rawint.txt) is used both in the MATLAB simulation file and the Visual Basic program running the FPGA computer board.

Processing Parameters: The processing parameters of the specified false target consist of a phase increment/decrement corresponding to the selected Doppler shift, and the gain coefficients representing the amplitude modulation.

Range and Azimuth compression: These parts represent basic signal processing functions in the ISAR. Pulse compression is performed on the radar return signal from the false target generated by the FPGA DIS.

Display: After processing, the signals will be presented to the user as an image. In this case it will be done by a series of plot using MATLAB (as described in Chapter 4).

FPGA DIS

Buffers: "Buffer 1" is for storage of DRFM phase data samples to be fed into the tapped delay lines. "Buffer 2" is for storage of modulation parameters, which are computed and updated by the host. These include parameters for target extent, amplitude modulation and Doppler shift. "Buffer 3" is for storing the outputs of the DIS (modulated signals).

Tap 0 to 2: Three tapped delay lines have been implemented using the FPGA technology in order to study the trade-offs involved. Each tap basically consists of a delay element, implemented in hardware using a cascaded chain of flip-flops. The phase adder together with the look-up table provides a Doppler modulated complex signal. The gain modules provide amplitude modulation to the signal, represented by the triangular symbols connected to the outputs of the look-up tables.

Summer: The summer adds outputs from I- and Q-channels separately together. The addition is done by first taking a partial sum of the outputs from two last taps, and then as another step adding this result to the output of the first tap.

The hardware used for the DIS implementation, and its interface with the host computer is shown in Figure 5.5 and Figure 5.6. Figure 5.5 shows a photo of the host computer, a PII 300 MHz with 128 MB RAM. Figure 5.6 shows the DIS hardware consisting of a FPGA (Altera 10K50 FPGA chip) mounted on a Naval Research Laboratory (NRL) custom designed computer board and can be seen inserted in the lower slot of the computer. The Altera 10K50 FPGA chip is the large device in the center of the board.

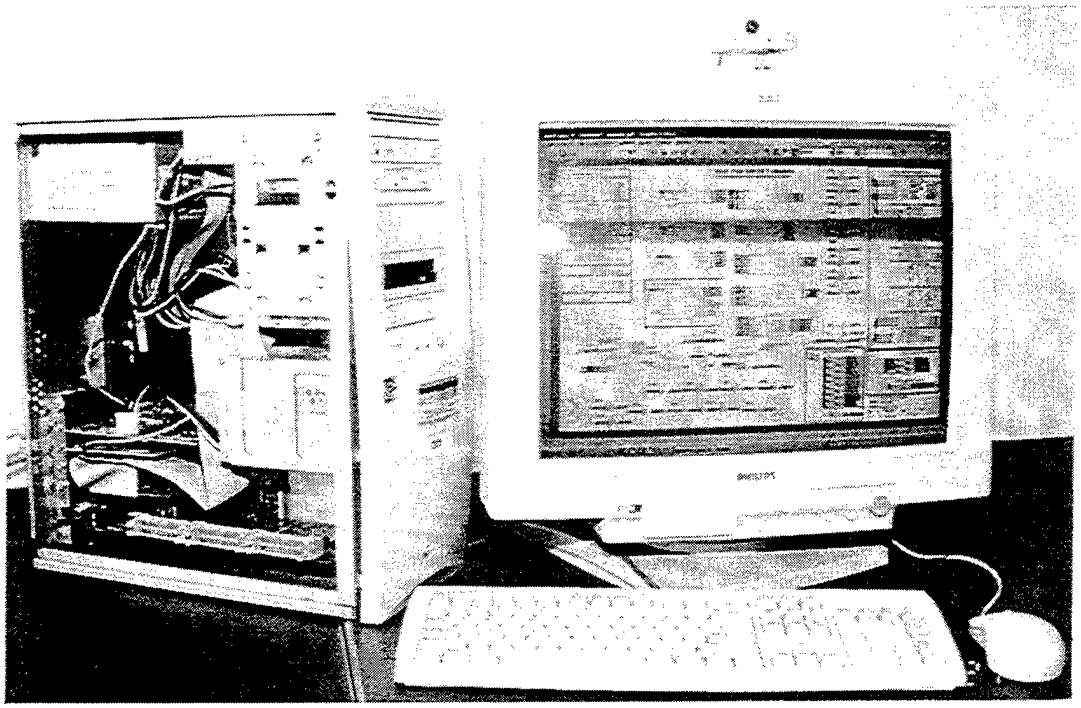


Figure 5.5: Picture of the concept demonstrator -- host (PC) with FPGA computer board (DIS).

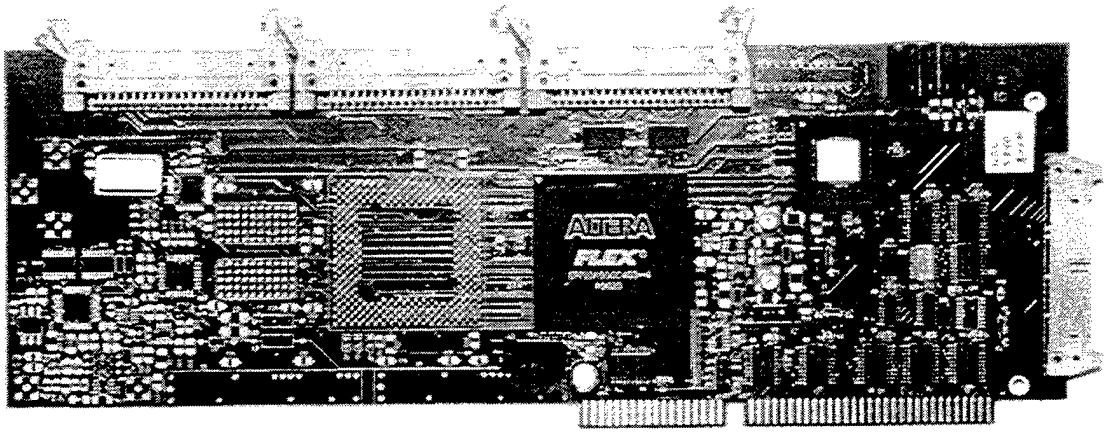


Figure 5.6: Picture of the customized 1-FPGA computer board used for the DIS prototype.

Processing of DRFM phase data samples by the three-tap original DIS architecture can be visualized as follows. For each received radar chirp pulse, a set of phase samples will be

provided by the DRFM. At startup, valid output data consists of only the output from Tap 0. At the next clock cycle valid data will be the sum of processed data from Tap 0 and Tap 1. At the third clock cycle the output will be the sum of processed data from all three taps. At the end of the pulse, the taps are shutdown in reverse order while the phase data is propagating through the delays. An example of 64 radar pulses and 62 DRFM phase data samples (range bins) per radar pulse are summarized in Table 5.4.

Radar Pulse	DRFM Phase Data	Clk	Tap 0	Tap 1	Tap 2	Result
1	D_1	0	$P_n(D_1)$	0	0	$P_n(D_1)$
1	D_2	1	$P_n(D_2)$	$P_{n+1}(D_1)$	0	$P_n(D_2) + P_{n+1}(D_1)$
1	D_3	2	$P_n(D_3)$	$P_{n+1}(D_2)$	$P_{n+2}(D_1)$	$P_n(D_3) + P_{n+1}(D_2) + P_{n+2}(D_1)$
.	.	.	.	.	.	
.	.	.	.	.	.	
1	D_{62}	61	$P_n(D_{62})$	$P_{n+1}(D_{61})$	$P_{n+2}(D_{60})$	$P_n(D_{62}) + P_{n+1}(D_{61}) + P_{n+2}(D_{60})$
1	-	62	0	$P_{n+1}(D_{62})$	$P_{n+2}(D_{61})$	$P_{n+1}(D_{62}) + P_{n+2}(D_{61})$
1	-	63	0	0	$P_{n+2}(D_{62})$	$P_{n+2}(D_{62})$
.	.	.	.	.	.	
2	D_1	64	$P_n(D_1)$	0	0	$P_n(D_1)$
2	D_2	65	$P_n(D_2)$	$P_{n+1}(D_1)$	0	$P_n(D_2) + P_{n+1}(D_1)$
2	D_3	66	$P_n(D_3)$	$P_{n+1}(D_2)$	$P_{n+2}(D_1)$	$P_n(D_3) + P_{n+1}(D_2) + P_{n+2}(D_1)$
.	.	.	.	.	.	
.	.	.	.	.	.	
64	D_{62}	4093	$P_n(D_{62})$	$P_{n+1}(D_{61})$	$P_{n+2}(D_{60})$	$P_n(D_{62}) + P_{n+1}(D_{61}) + P_{n+2}(D_{60})$
64	-	4094	0	$P_{n+1}(D_{62})$	$P_{n+2}(D_{61})$	$P_{n+1}(D_{62}) + P_{n+2}(D_{61})$
64	-	4095	0	0	$P_{n+2}(D_{62})$	$P_{n+2}(D_{62})$

Table 5.4: Correct Processing of DRFM Phase Samples (Original DIS Architecture).

Remarks concerning Table 5.4

Notation	*Description*
Radar Pulse	Represents one radar pulse. The number of radar pulses represents the number of Doppler cells for the ISAR, in this case 64.
DRFM Phase Data	62 DRFM phase samples per radar pulse in this specific case.
Clk	Clock pulse for the DIS.
Tap n	Output of the n^{th} tap.
Tap n+1	Output of the (n+1) tap.
Tap n+2	Output of the (n+2) tap (the last tap in this example).
Result	The output from the DIS.
$P_{n+x}(D_x)$	Processed phase data in a tap available as valid output.

The processing of DRFM phase data in the three taps that has been implemented in hardware using FPGA technology is shown in Table 5.5. It must be noted that the implementation of the DIS algorithm using FPGAs does not perform a correct startup and shutdown of the individual taps when a set of DRFM phase samples is processed. Instead a data value of zero is processed through the tap and produces an incorrect output due to the cosine look-up table (*cos(0) = 1*). This adds a slight error at the beginning and trailing edges of the pulse when compared with the MATLAB simulation that strictly follows the original DIS algorithm.

Radar Pulse	*DRFM Phase Data*	*Clk*	*Tap n*	*Tap n+1*	*Tap n+2*	*Result*
1	D_1	0	$P_n(D_1)$	$P_{n+1}(0)$	$P_{n+2}(0)$	$P_n(D_1) + P_{n+1}(0) + P_{n+2}(0)$
1	D_2	1	$P_n(D_2)$	$P_{n+1}(D_1)$	$P_{n+2}(0)$	$P_n(D_2) + P_{n+1}(D_1) + P_{n+2}(0)$
1	D_3	2	$P_n(D_3)$	$P_{n+1}(D_2)$	$P_{n+2}(D_1)$	$P_n(D_3) + P_{n+1}(D_2) + P_{n+2}(D_1)$
.	.	.	.	.	.	
.	.	.	.	.	.	
1	D_{62}	61	$P_n(D_{62})$	$P_{n+1}(D_{61})$	$P_{n+2}(D_{60})$	$P_n(D_{62}) + P_{n+1}(D_{61}) + P_{n+2}(D_{60})$
1	0	62	$P_n(0)$	$P_{n+1}(D_{62})$	$P_{n+2}(D_{61})$	$P_n(0) + P_{n+1}(D_{62}) + P_{n+2}(D_{61})$
1	0	63	$P_n(0)$	$P_{n+1}(0)$	$P_{n+2}(D_{62})$	$P_n(0) + P_{n+1}(0) + P_{n+2}(D_{62})$
1	0	64	$P_n(0)$	$P_{n+1}(0)$	$P_{n+2}(0)$	$P_n(0) + P_{n+1}(0) + P_{n+2}(0)$
.	.	.	.	.	.	
2	D_1	64	$P_n(D_1)$	$P_{n+1}(0)$	$P_{n+2}(0)$	$P_n(D_1) + P_{n+1}(0) + P_{n+2}(0)$
2	D_2	65	$P_n(D_2)$	$P_{n+1}(D_1)$	$P_{n+2}(0)$	$P_n(D_2) + P_{n+1}(D_1) + P_{n+2}(0)$
2	D_3	66	$P_n(D_3)$	$P_{n+1}(D_2)$	$P_{n+2}(D_1)$	$P_n(D_3) + P_{n+1}(D_2) + P_{n+2}(D_1)$
.	.	.	.	.	.	
.	.	.	.	.	.	
64	D_{62}	4096	$P_n(D_{62})$	$P_{n+1}(D_{61})$	$P_{n+2}(D_{60})$	$P_n(D_{62}) + P_{n+1}(D_{61}) + P_{n+2}(D_{60})$
64	0	4097	$P_n(0)$	$P_{n+1}(D_{62})$	$P_{n+2}(D_{61})$	$P_n(0) + P_{n+1}(D_{62}) + P_{n+2}(D_{61})$
64	0	4098	$P_n(0)$	$P_{n+1}(0)$	$P_{n+2}(D_{62})$	$P_n(0) + P_{n+1}(0) + P_{n+2}(D_{62})$
64	0	4159	$P_n(0)$	$P_{n+1}(0)$	$P_{n+2}(0)$	$P_n(0) + P_{n+1}(0) + P_{n+2}(0)$

Table 5.5: Processing of DRFM Phase Samples Using FPGAs (Original DIS Architecture).

Remarks concerning Table 5.5

Notation	*Description*
Radar Pulse	Represents one radar pulse. The number of radar pulses represents the number of Doppler cells for the ISAR, in this case 64.
DRFM Phase Data	62 DRFM phase data samples per radar pulse in this specific case.
Clk	Clock pulse for the DIS.
Tap n	Output of the n^{th} tap.
Tap n+1	Output of the (n+1) tap.
Tap n+2	Output of the (n+2) tap (the last tap in this example).
Result	The output from the DIS.
$P_{n+x}(D_x)$	Processed phase data sample in a tap available as valid output.
$P_{n+x}(0)$	Processed "0" in a tap available as output.

Top-Level FPGA Hierarchy: The top-level hierarchy of the design using FPGAs is shown in Figure 5.7. The bottom left hand block is the I/O-decode and Built-in-Test (BIT) block. The purpose of the I/O decode block is to provide up to 256 addressable "internal" address spaces for reading and writing. The other blocks have direct correspondence to the other modules in the DIS:

- Tap-delay line (*delay.gdf*)
- Doppler modulation coefficient latch (*phi.gdf*)
- Phase summer (*ph_acc.gdf*)
- Look-up-table (*lut.gdf*)
- Gain modulation coefficient latch (*gain.gdf*)
- Gain modulator (*newgain1.gdf*)
  - (*shift0.gdf, shift1.gdf, shift2.gdf*)
    - (*mux2.gdf*)
- Output summer (*out_summer.gdf*)

Each of these modules are described in further detail below.

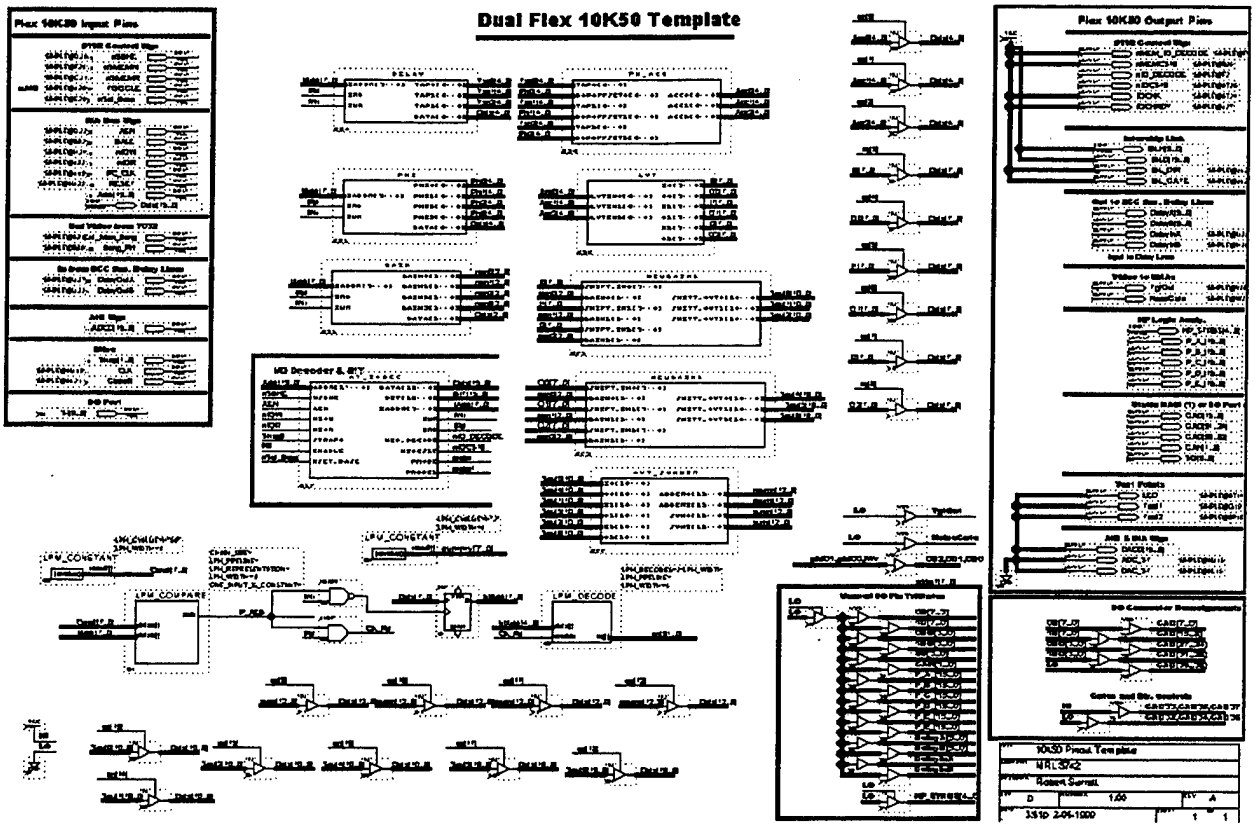


Figure 5.7: Top-level FPGA hierarchy of the DIS (simple.gdf).

Tap-Delay Line: The tap-delay line schematic with 3 tap delay lines is shown in Figure 5.8. The tap-delay lines are made up of a chain of D-flip-flops and occupy 4 internal addresses, 0x30, 0x31, 0x32 and 0x33. The meaning of the data values written to these locations is described in Table 5.6.

Internal Address (in hex)	*Function*
0x30	Write "1" to reset tap-delay line, "0" otherwise.
0x31	Write any value to this address to cause a propagation of the values down the delay line.
0x32	Write the new DRFM value to the first tap of the delay line.
0x33	Unused.

Table 5.6: Internal Address Usage in the Tap Delay Line

Updating the tap-delay line is a 2-stage process. This is accomplished by writing any value into address 0x31 (to effect propagation) followed by writing a new value into address 0x32 (to load in a new value at the first tap of the delay line).

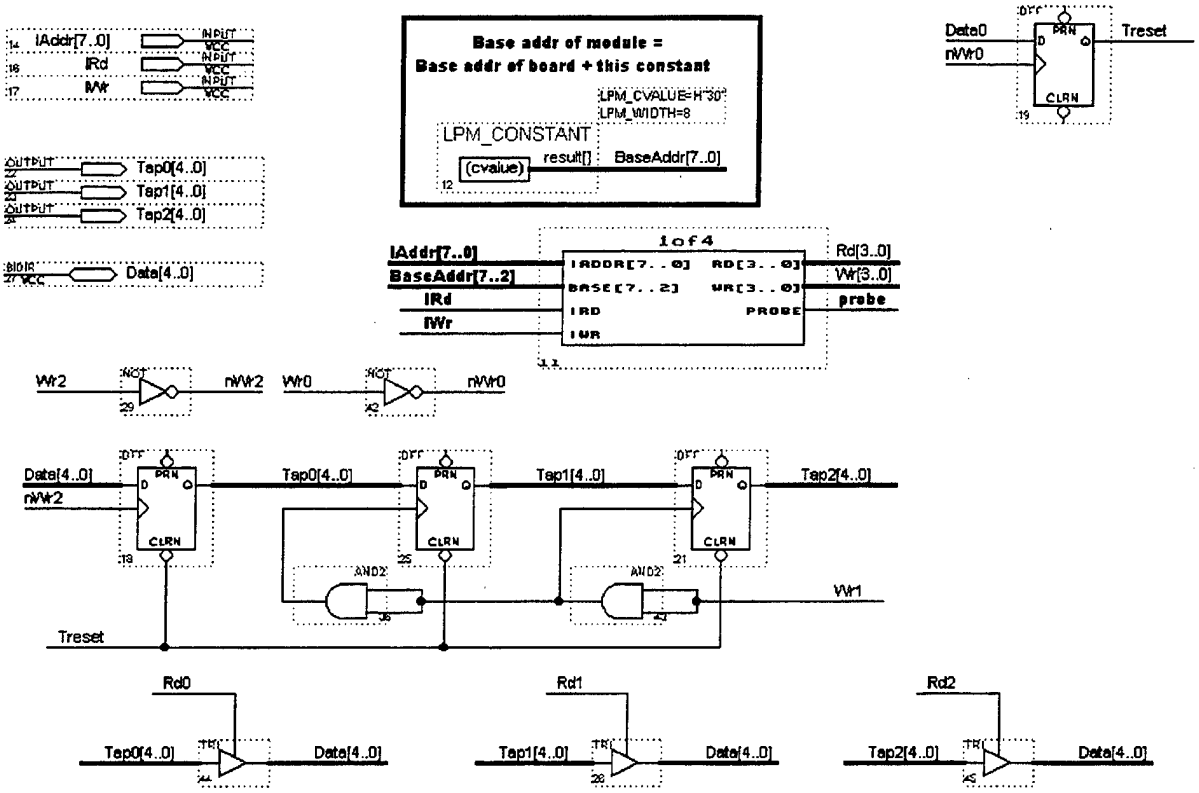


Figure 5.8: Schematic of the tap-delay line (delay.gdf).

Doppler Modulation Coefficient Latch: The phase coefficient latch (for Doppler modulation) is comprised of a 1-of-4 decoding block and a set of flip-flops as shown in Figure 5.9.

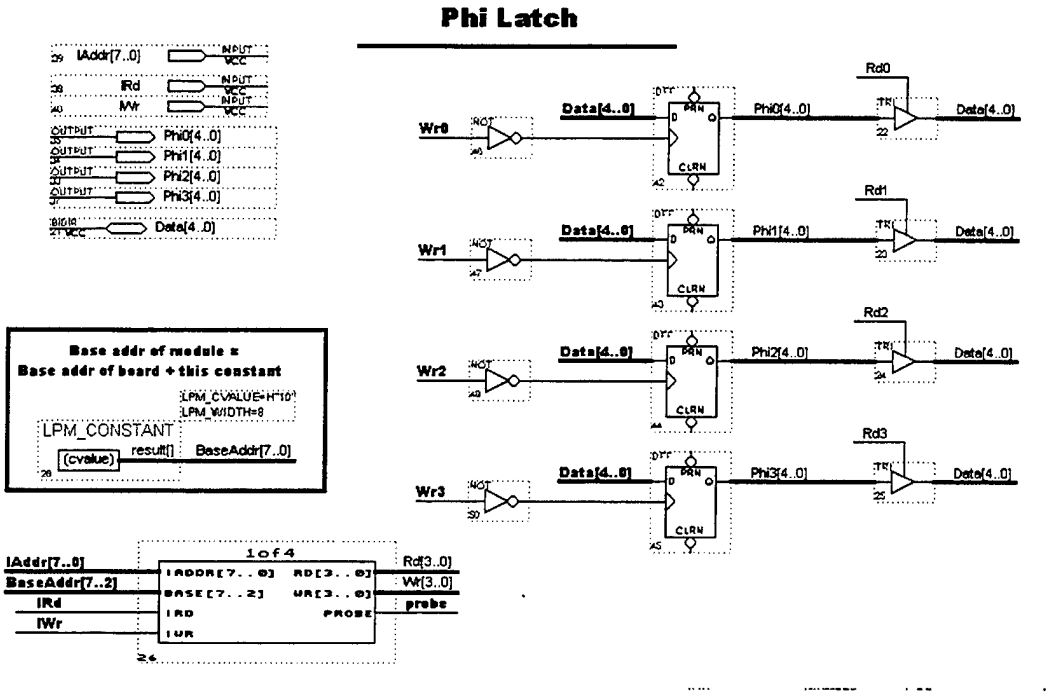


Figure 5.9: Schematic of the phase coefficient latch for Doppler modulation (phi.gdf).

Phase Accumulator: The phase accumulator schematic is shown in Figure 5.10 (one for each tap). The inputs to the accumulator are the 5-bit DRFM phase samples (the values from the tap-delay line) and the latched 5-bit phase coefficients. Furthermore, the output bit-width matches the input bit-width (the carry-bits are discarded) representing a modulus addition operation (which is desired). Due to truncation of values larger than 5 bits, the phase values above 2π are folded back into the principle range between 0 and 2π. The LPM-ADD-SUB module available in the Library of Parameterized Module (LPM) is used to configure the adder.

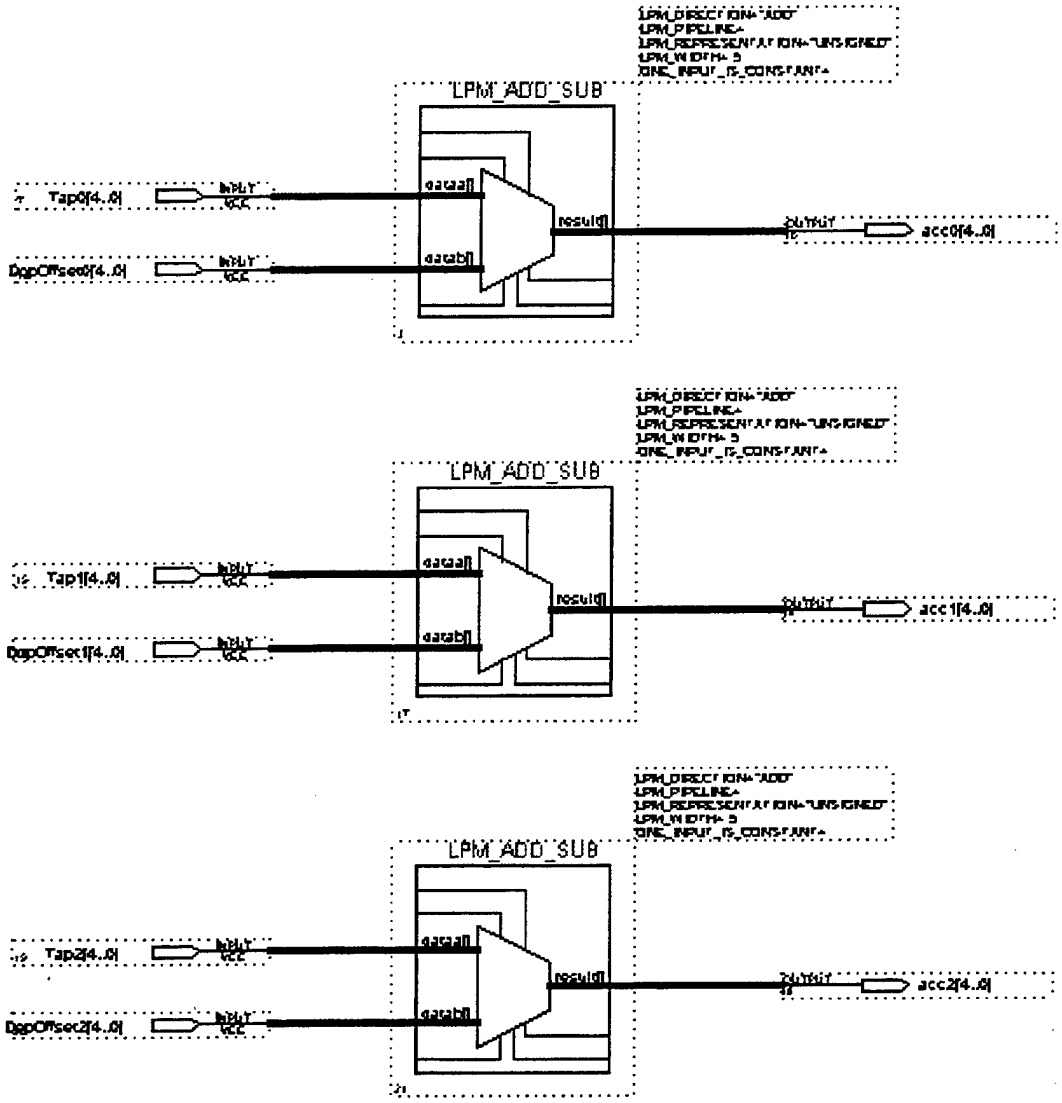


Figure 5.10: Schematic of the phase accumulator (ph_acc.gdf).

Look-Up Table (LUT): The look-up table (LUT) is indexed by the output phase from the phase accumulator. This phase value is mapped to an 8-bit amplitude value stored in the LUT. Since the LUT output is a complex number, a cosine and sine table indexed by the same phase is required. The schematic diagram for the LUT is given in Figure 5.11. For the LUT configuration, a text file is associated with each LPM_ROM module. In Altera, this file is called a memory initialization file. A MATLAB script file (*genLUT.m*) which is capable of automatically generating the text-file, based on the width and depth of the LUT desired has been used. This file generates the memory initialization file for the LPM_ROM module (*cos.mif* and *sin.mif*). It calls two MATLAB function files (*genfixptv0.m* and *genfloat.m*). These two programs perform the fix and floating point conversions and are included in the Appendix.

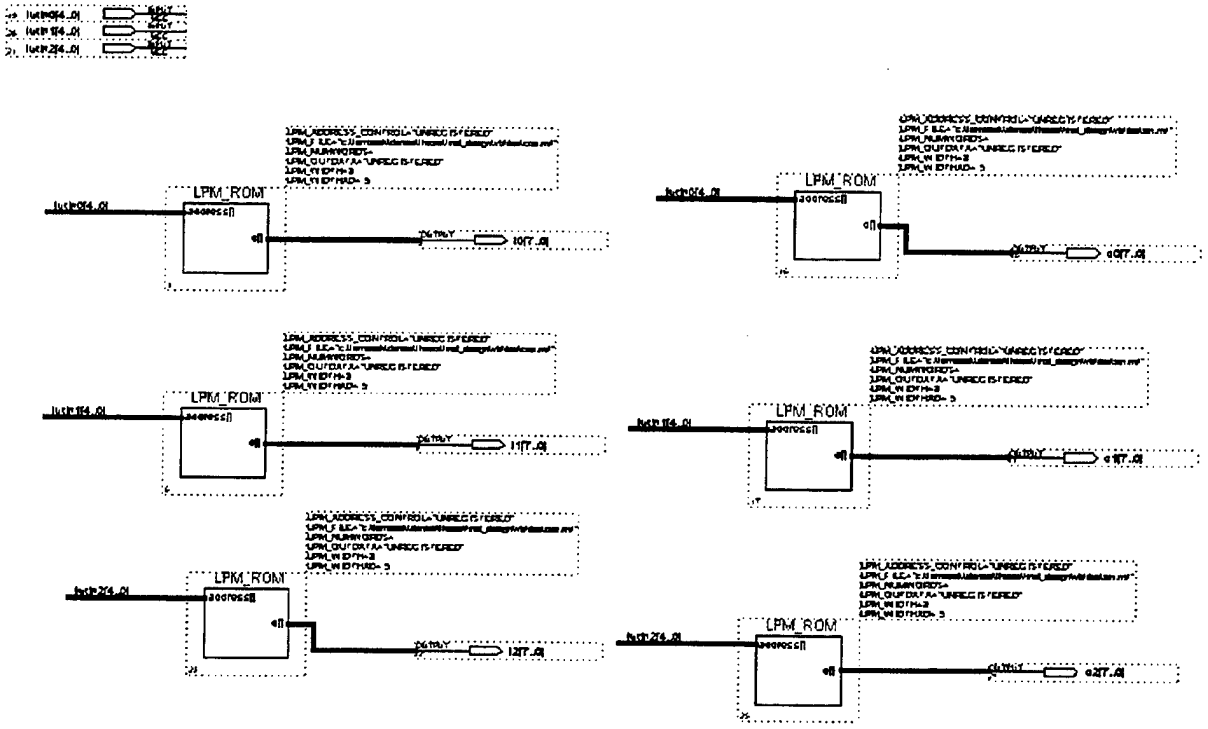


Figure 5.11: Schematic diagram of the look-up table (LUT) (lut.gdf).

Gain Modulation Coefficient Latch: The latch for the gain modulation coefficient comprises a 1-of-4 decoding block and a set of flip-flops as shown in Figure 5.12. Although four DQ flip-flops are shown, only three of them are used (one for each tap).

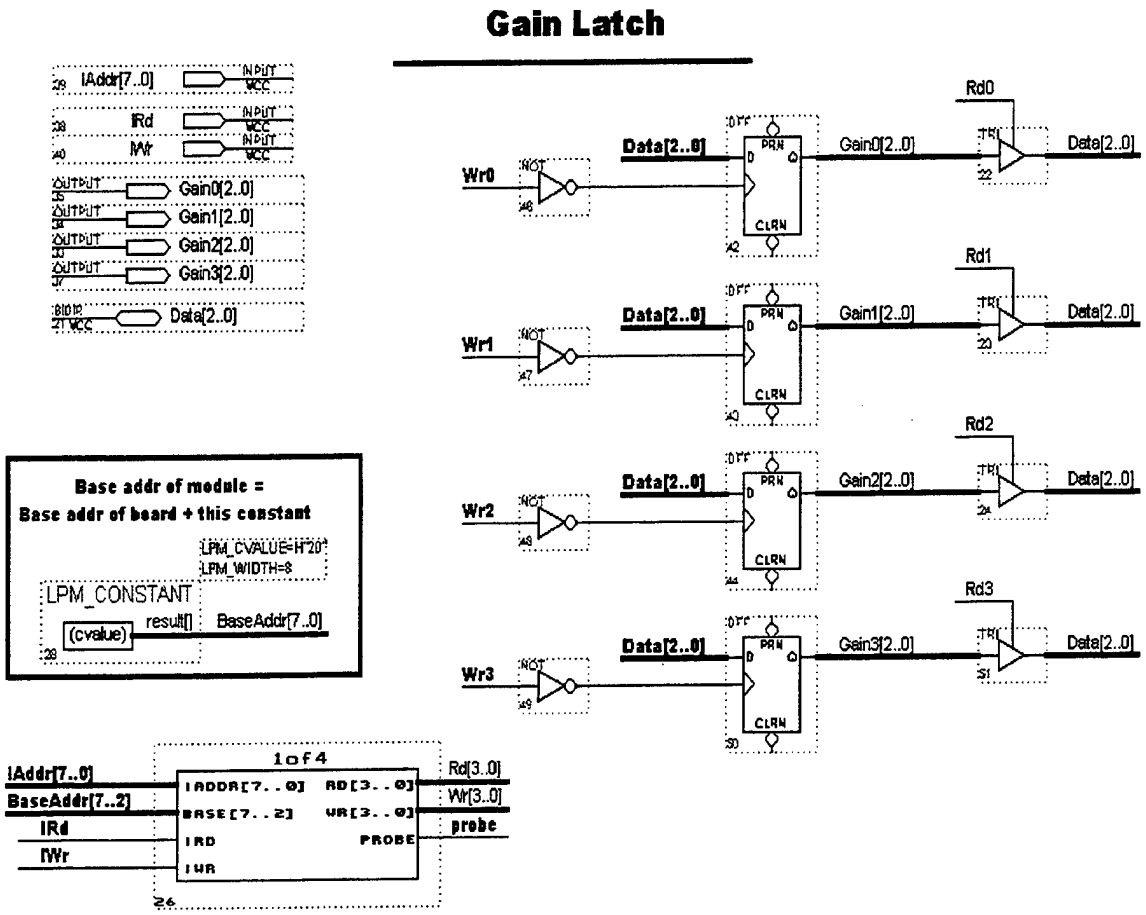


Figure 5.12: Schematic of the gain modulation coefficient latch (gain.gdf).

Gain Modulator: The gain modulator applies a gain to the binary signal from the LUT by shifting the binary word towards the most significant bit position and pads zeros at the least significant bit position. The gain modulator is shown in Figure 5.12.

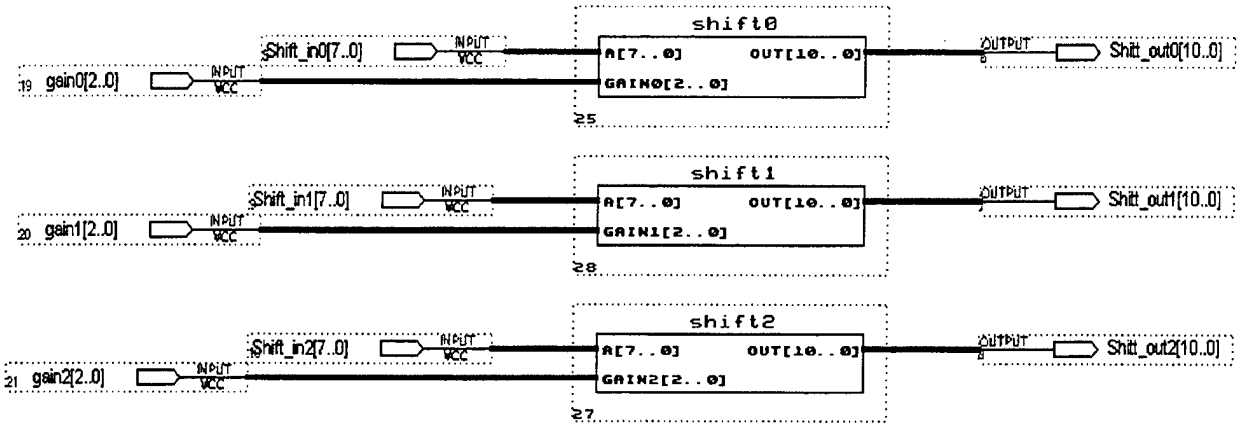


Figure 5.12: Schematic of the gain modulation (newgain1.gdf).

The original amplitude values, as set by the user in the MATLAB GUI (the Range-Doppler-Amplitude Map Entry Program), are "translated" into a corresponding number of positions for the shift according to Table 5.7.

GUI Amplitude Value	*# of Shift Left Steps*	*Represents decimal multiplication by*
1	0	1
2	1	2
3	2	4
4	3	8

Table 5.7: Translation of Gain Values

Figure 5.13 exemplifies the results of applying different gain modulation coefficients. In subplot 1 a GUI Amplitude value of "1" was applied, representing a decimal gain value of "1", for a 3-target cell long target. In subplot 2 to 4 the GUI Amplitude value was increased to 2, 3, and 4 respectively, representing a decimal gain value of 2, 4, and 8.

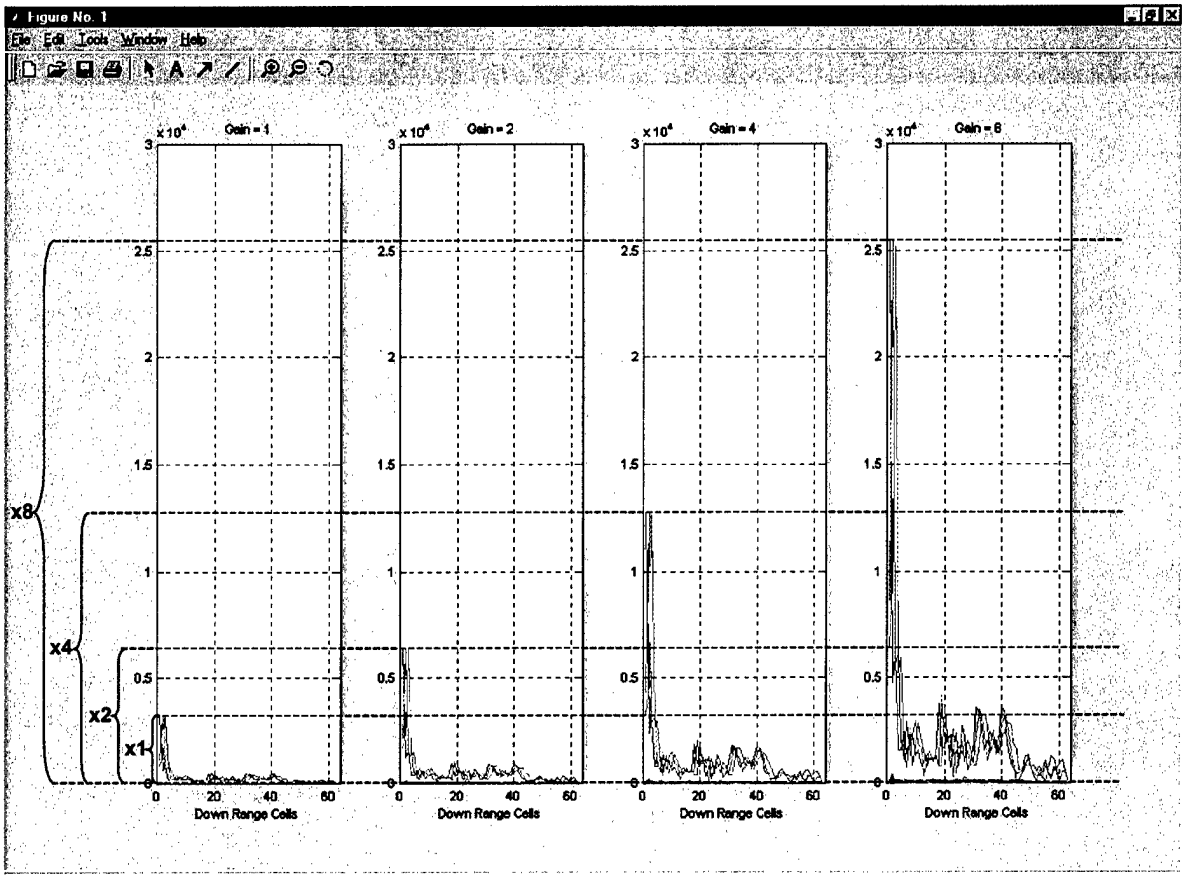


Figure 5.13: A 3-Target Cell long target, with different gain modulation coefficients.

The schematic diagram of the shift primitive is shown in Figure 5.14 and the schematic diagram of the MUX2 building block is provided in Figure 5.15.

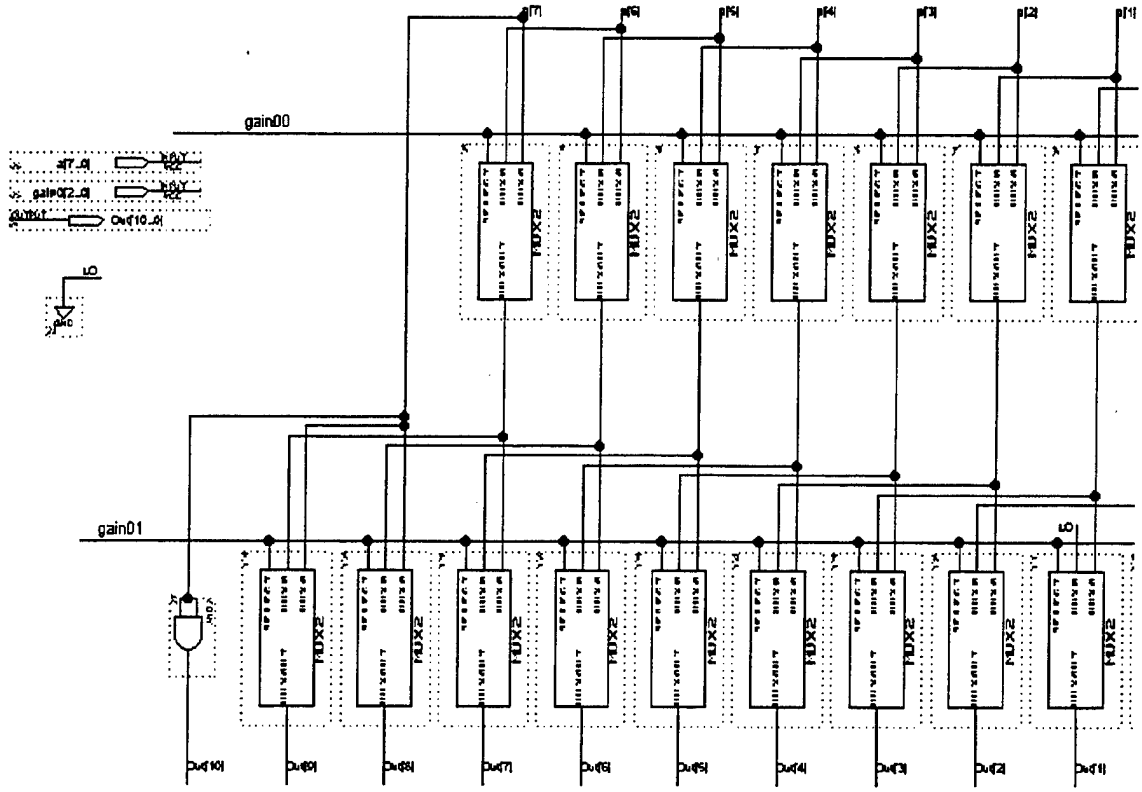


Figure 5.14: Schematic of the shift primitive in the gain modulation block (shift0.gdf).

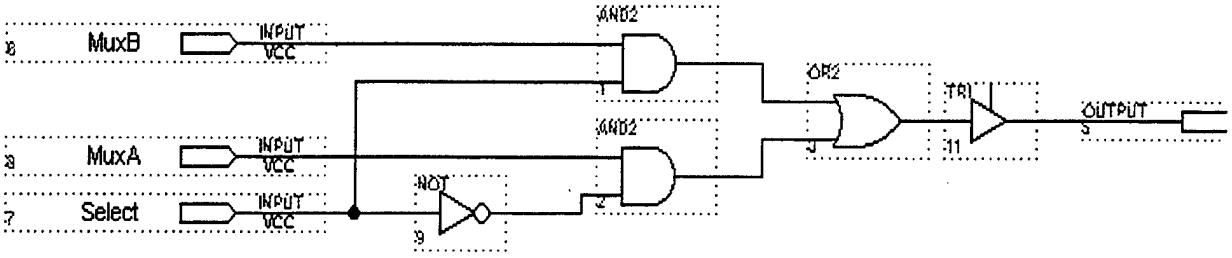


Figure 5.15: Schematic of the MUX2 in the gain modulation block (mux2.gdf).

Final Summer: The schematic of the final summer is given in Figure 5.15. This circuit implements the addition of the tap outputs in two's-complement. The addition in twos-complement involves sign-extension of the numbers to be added and the discarding the carry out bit. The LPM-ADD-SUB module, available in the LPM, is used to configure the summer.

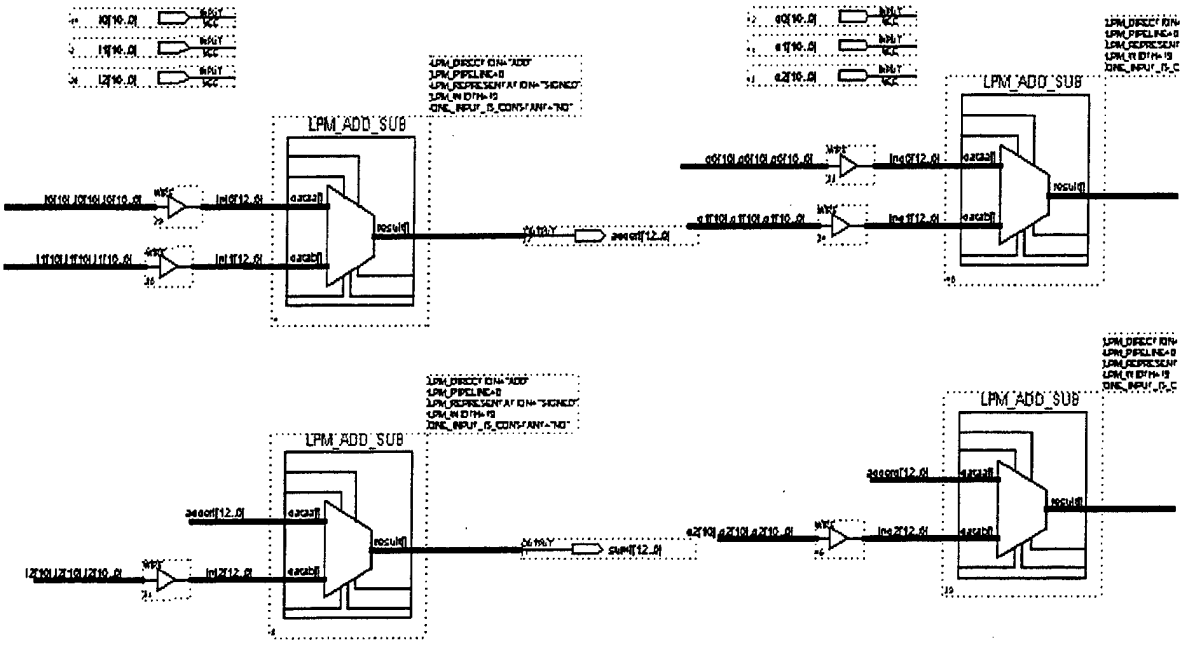


Figure 5.16: Schematic of the final summer (out_summer.gdf).

E. Simulation Results

Simulation Setup: Several simulations have been done to verify that the results are as expected. Below is one example of a simulation run to illustrate the steps and to visualize the results. In this case the false target to be generated has the same parameters given in the example above (MATLAB simulation) as is shown again in Figure 5.17.

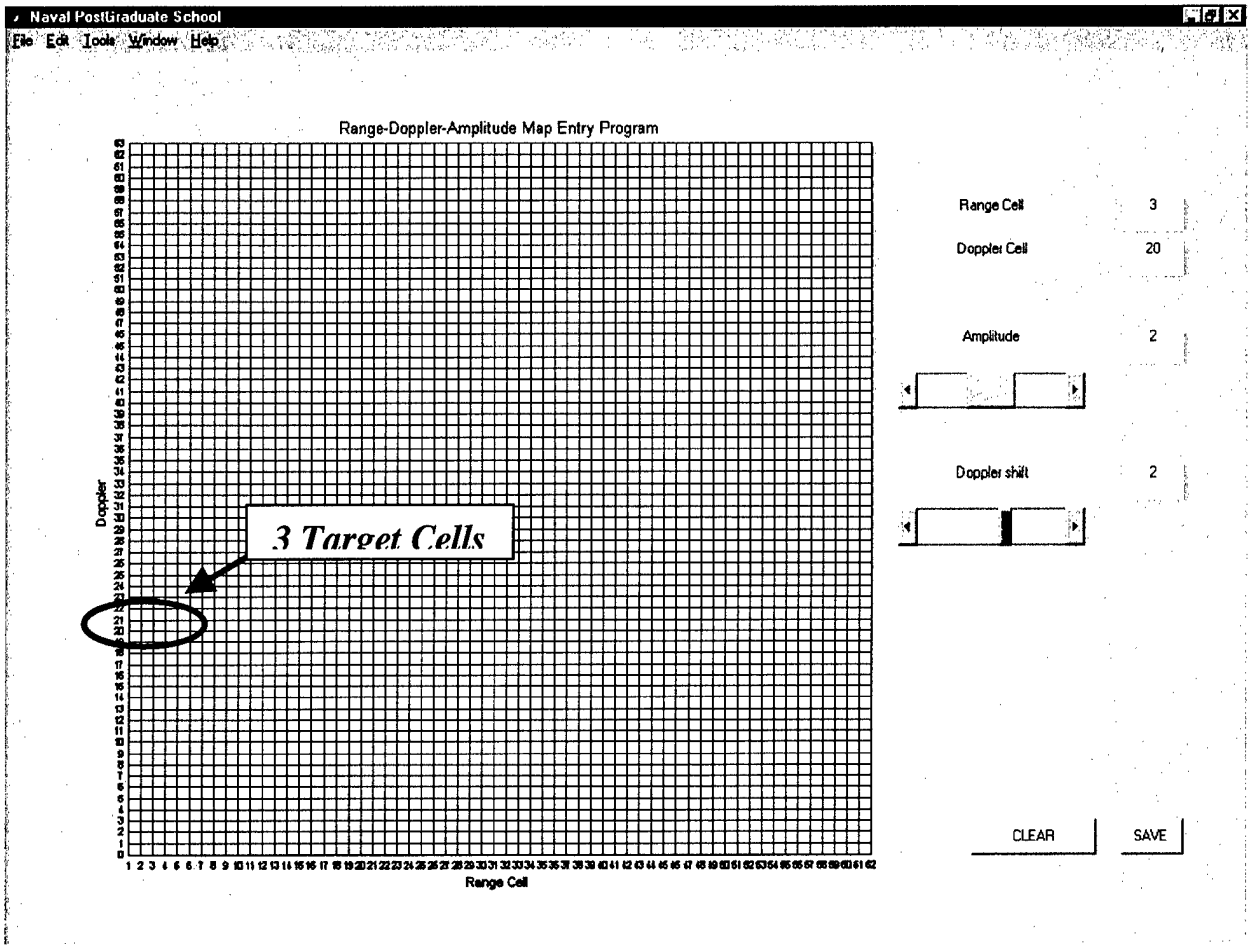


Figure 5.17: The range-Doppler-amplitude map entry program.

In this example the user has specified the data in Table 5.8 for the false target to be created by the DIS.

Target Cell	*Range Cell*	*Doppler Cell*	*Amplitude*	*Doppler Shift*	*Remark*
1	1	20	2	0	Tap 0 – 1st Tap
2	2	20	2	1	Tap 1 – 2nd Tap
3	3	20	2	2	Tap 2 – 3rd Tap

Table 5.8: User Specified Inputs of the False Target

To be able to make the comparison between the MATLAB simulation and the DIS implemented using FPGA technology, an intermediate step was added in the simulation flow as described in Chapter 4. After the MATLAB file *mathostvX.m* has been executed, all necessary inputs are available in text files to run the hardware implementation of the DIS. The interface with the FPGA computer board is a set of Visual Basic files composed into a Visual Basic project called FlexTest (*flextest.vbp*). To be able to compile and run the project and the board properly the necessary files have to be located in a file structure with the following path: *c:\temasek\denise\thesis\final_design\vbfiles*. To run the Visual Basic project FlexTest the user must open the project, open the *the_isar.bas* file, and the run the file. Another GUI will show up on the computer display to visualize the signal processing taken place in the taps of the DIS.

Simulation Results: The 2-D contour plots in Figure 5.18 show the results from the MATLAB simulation and the results from running the DIS implemented on the Altera FPGA device.

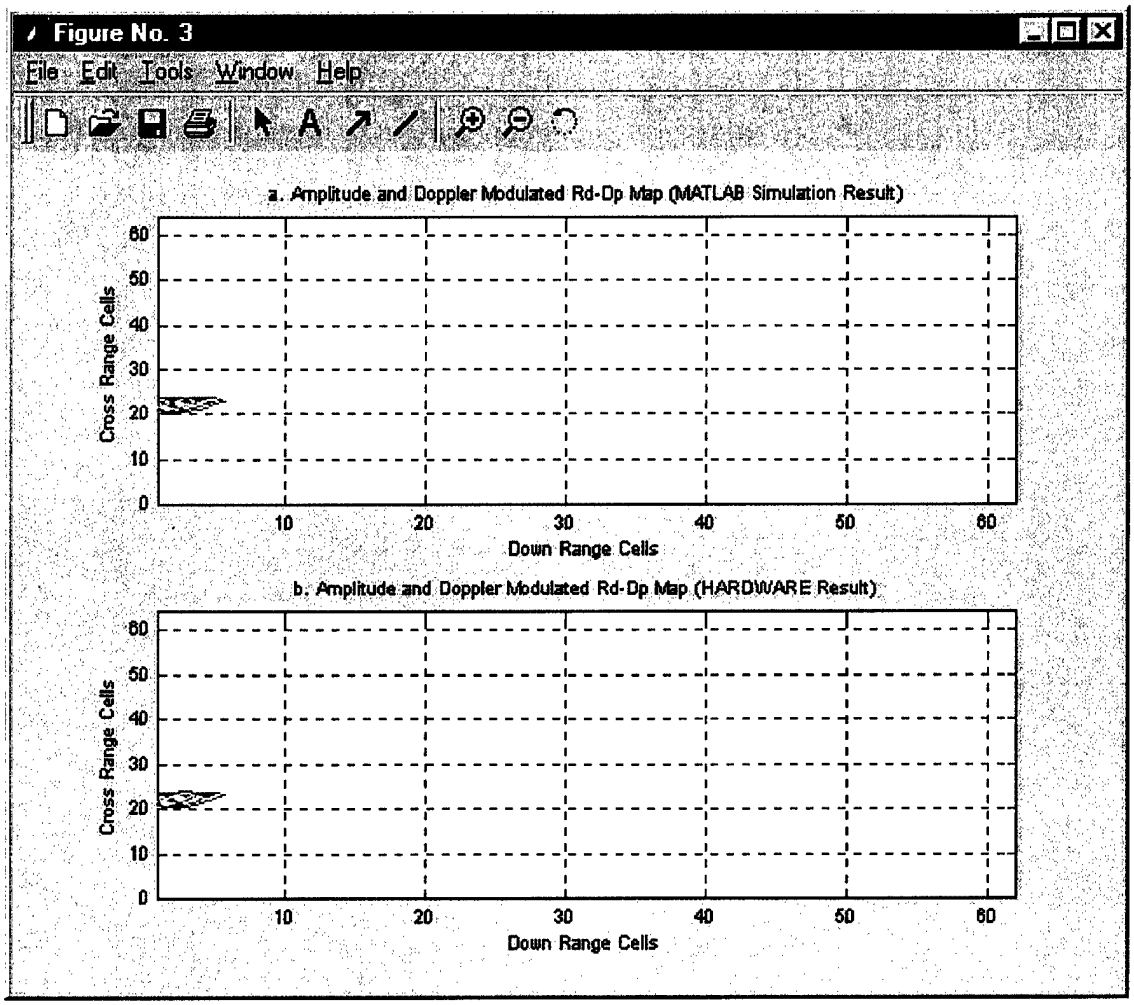


Figure 5.18: MATLAB DIS simulation versus FPGA hardware results.

Figure 5.19 shows the 3-D mesh surface plots. The first sub-plot shows the results from the MATLAB DIS simulation. The second shows the result from the FPGA hardware. Finally, the third sub-plot shows the difference between the MATLAB simulation and the FPGA hardware results.

As expected, a slight difference between the MATLAB simulation and the hardware results can be observed (note the scale on the amplitude axis of the three sub-plots). These differences are due to the fact that the implementation of the DIS algorithm using FPGAs do not consider a correct startup and shutdown of the individual taps when a set of DRFM phase data from one radar pulse is processed (as discussed earlier). This contributes to a slight error in comparison with the MATLAB simulation (that strictly follows the original DIS algorithm).

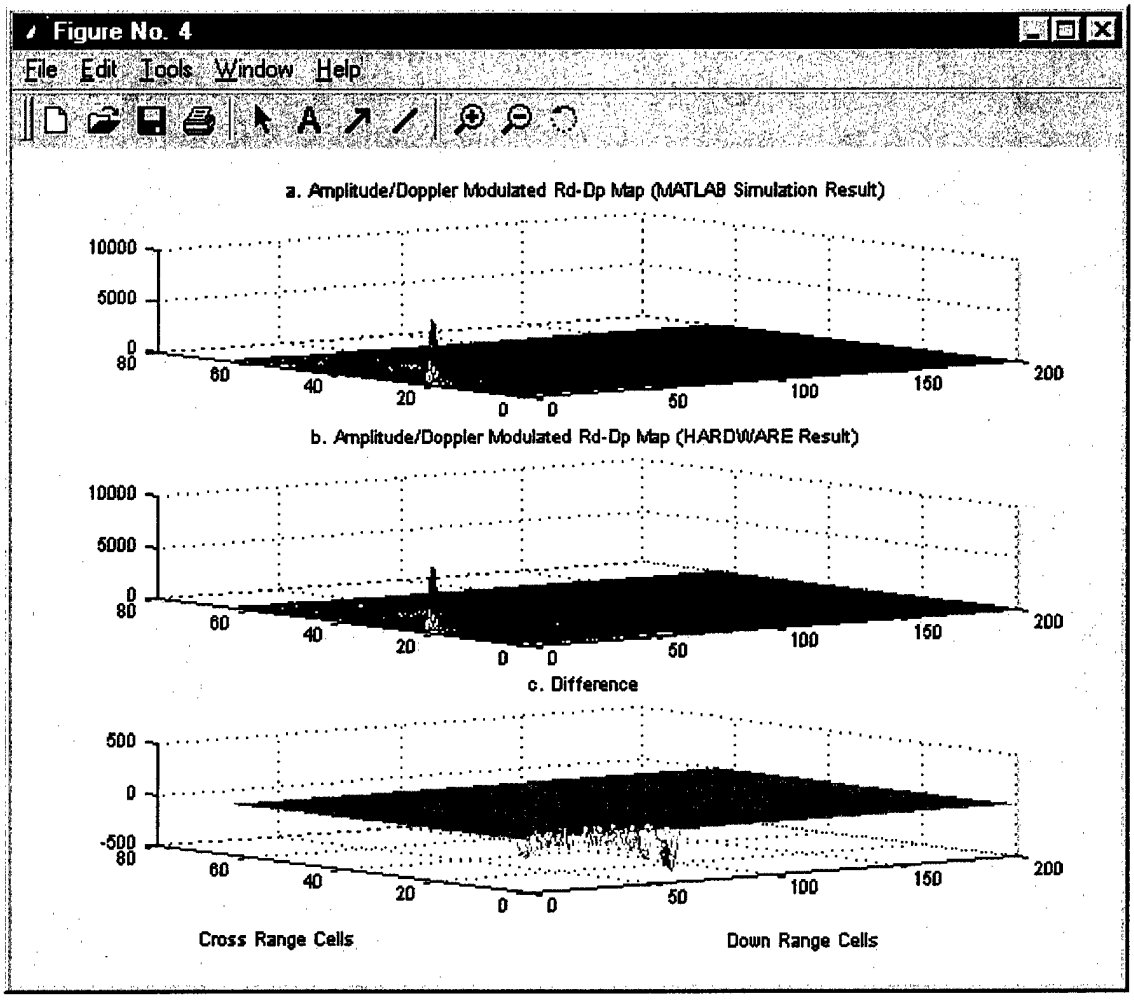


Figure 5.19: MATLAB simulation result versus FPGA hardware result and their difference.

To verify that the errors actually are due to the difference in startup and shutdown sequences the MATLAB simulation code was adjusted to process the phase data in the same manner as the FPGA hardware. The results of the modified test case are shown in Figure 5.20. As expected, there are no differences between the MATLAB simulation results and the hardware FPGA results.

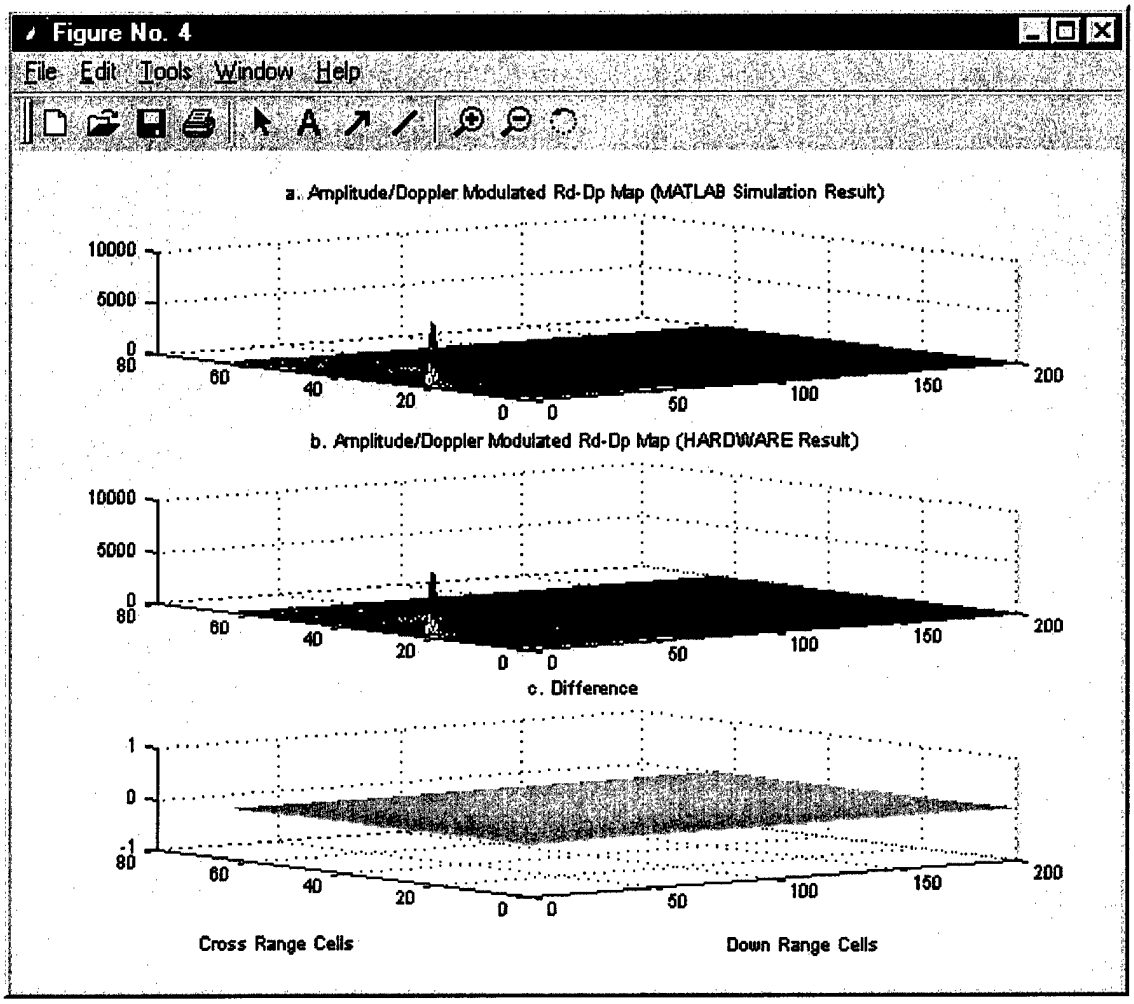


Figure 5.20: MATLAB simulation results versus FPGA hardware results and their difference.

6. FPGA-TO-ASIC CONVERSION

A. FPGA Limitations

After analyzing the original, nearly complete implementation of the original architecture, it was realized that there were several limitations being imposed on the design solely, because the implementation was done by using FPGA technology. First and foremost was the speed limitation. The target clock speed for the design is 2ns. This is an aggressive goal for any new chip design and although it might eventually be possible to meet this target with an FPGA design, in the foreseeable future, a full-custom IC has a higher probability of meeting this speed requirement. A second major contributing factor was the physical size of the implementation. The initial, proof-of-concept design does not require a large number of taps. However, even with a small number of taps the design could not be implemented into a single FPGA. One of the goals for this initial, proof-of-concept design was to create a design that is easily extendable to more taps. Extending the FPGA implementation to more taps would require a significant increase in the number of FPGAs. This was considered a major drawback of the FPGA implementation.

After realizing the limitations of the FPGA implementation, it was decided to convert the FPGA design to an ASIC design. Several FPGA-to-ASIC conversion techniques were investigated. This Chapter discusses the different conversion methods and their related problems. It concludes with a summary of the problems encountered and the reasons behind choosing the Tanner Tools environment.

B. Altera-to-MOSIS Process Flow

The Altera-to-MOSIS conversion process investigated, attempts to translate the design from Altera's MaxPlus+ II implementation to a high speed ASIC fabricated by MOSIS. It will be shown that this is very complex and that parts of the conversion process are unpredictable since some tools don't have a common interface.

Altera to Mosis link Overview: The flowchart shown in Figure 6.1 shows the complete link from the FPGA design in Max+Plus II, via the conversion into an ASIC design, to finally the chip fabrication. All components will be explained and described in this section.

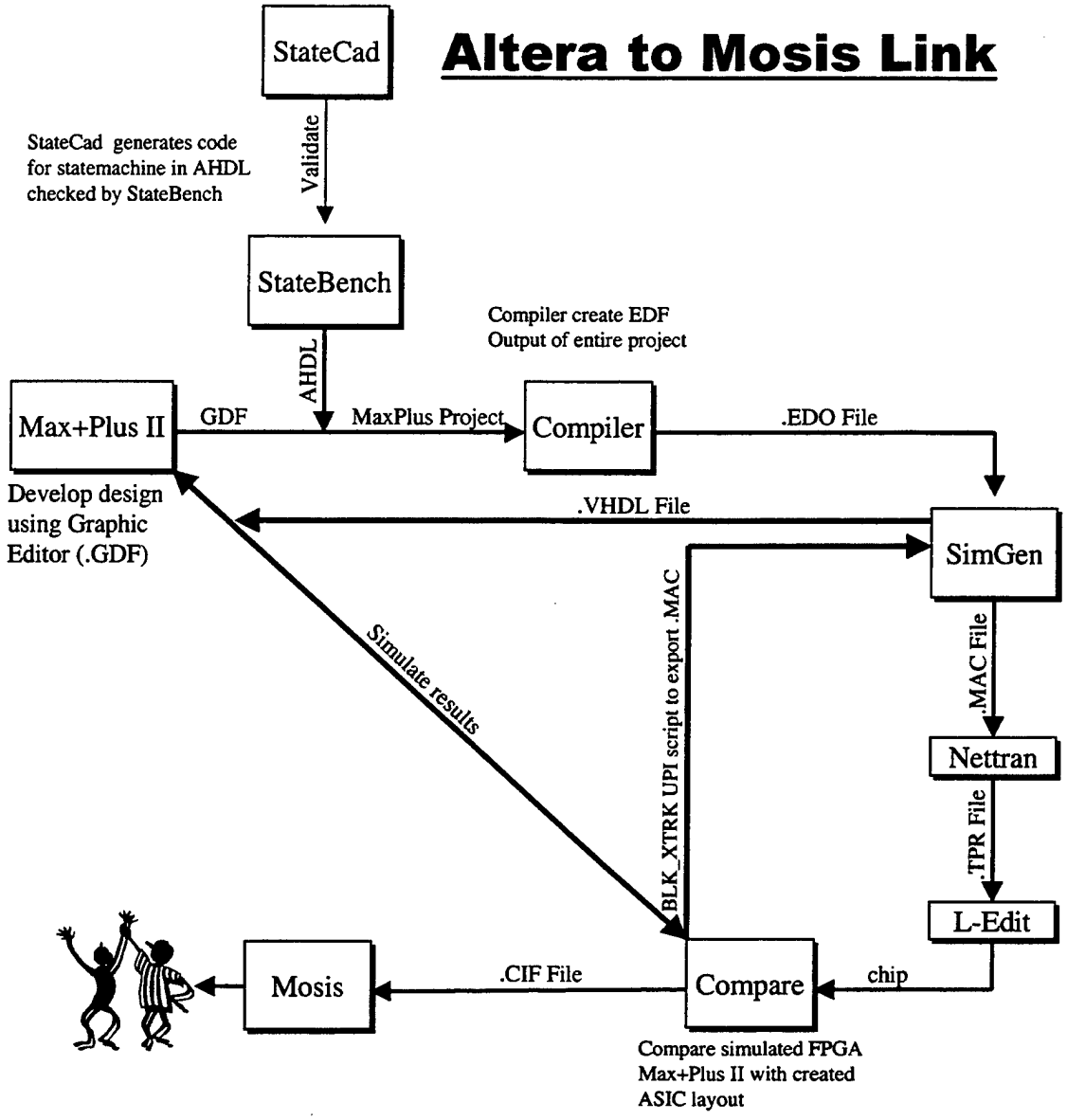


Figure 6.1: Flowchart – Altera to MOSIS link.

Remarks for Figure 6.1

GDF Graphic Design File is the file format of the Graphic Editor in Max+Plus II

AHDL Altera Hardware Description Language

EDO EDIF output file

MAC Macro file, to use between SimGen and Nettran

TPR Tanner Tools file type

CIF Chip fabrication format for the final layout

VHDL VHSIC Hardware Description Language

Program Descriptions :

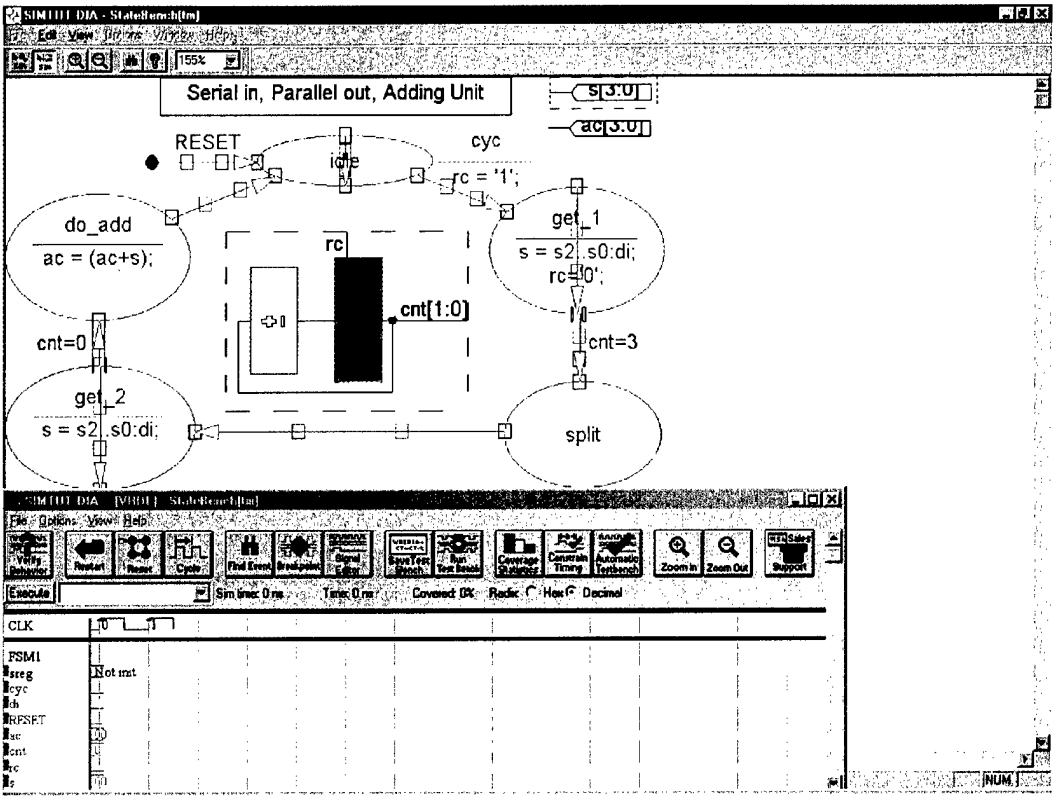


Figure 6.2: Statecad screenshot (from [10]).

Statecad: Statecad is a powerful tool to create state machines of all kinds in an easy way [11]. It is a graphical entry tool that allows the user to express ideas as state diagrams. Statecad has been designed for simplicity in use as a tool for digital design, documentation and error analysis. The Statecad GUI is shown in Figure 6.2. After validating a diagram, the program generates simulatable and synthesizable hardware description language (HDL) code directly from the

diagram. The HDL is valid, consistent, maintainable, and implements the graphical diagram. The HDL can be VHDL-1076, Verilog, ABEL-HDL, AHDL or ANSI-C. Interactive dialog boxes provide an environment for intuitive work and help to eliminate syntax errors and incomplete portions of state diagrams.

Once a design is completed in Statecad, it can be verified in the add-on software Statebench. After verification, a timing test bench can be written automatically. The test bench can be used for post synthesis timing verification.

Statebench is an add-on program to Statecad and automates behavioral verification and VHDL/Verilog test bench generation of any state diagram created in Statecad. Automatic test benches attempt to exercise every input, output, transition, and logic equation in a design. For verification the program can usually check approximately 80% of the design automatically. The remaining 20% require minor user inputs to complete the validation of the design. Statebench can generate VHDL or Verilog test benches for post-synthesis verification by adding time constraints that can be imported in third party test programs for further validation.

SimGen: SimGen is an EDIF to Nettran (Tanner Tools) and a FPGA to ASIC conversion utility for Tanner Tools EDA that improves routine operation designing within the Tanner environment. SimGen uses .EDIF or .MAC files as input and can generate VHDL files from a chip layout to support verifications in VHDL design flows. It automatically creates simulation files (.SIM and .VEC) for GateSim. For these types of files SimGen sets up template files with input/output lists and restores true port names. Due to its ability to create .MAC files, it supports file import into Tanner's Nettran software. Since the conversion between different file formats is not unproblematic, it attempts to clean up and repair netlists so they will work as expected when going from one tool to another. SimGen provides a Windows control shell to activate, coordinate, and generate command files for Tanner's remaining DOS tools, as well as file editing functions and waveform viewing functions [12].

Conclusion: The conversion process from Max+Plus II is long and in parts unpredictable, since SimGen has no direct supported interface to Max+Plus II. One major issue is the incompatibility between the library cells used in the FPGA design and the required library cells for an ASIC design. A significant amount of hand conversion of library cells is required. This is time consuming and potentially error prone. Other problems with this conversion approach included the efficiency of the conversion process with respect to speed, layout area, and power

consumption of the final IC design, and having to do future chip expansions using FPGA tools and then performing additional design conversions.

C. Leonardo Spectrum

An alternative program to create an ASIC can be found in Spectrum's Leonardo software [13]. Because of the very complicated creation of a workspace between Max+Plus II and Spectrum's software, as well as the missing capability to directly import file types, which are generated by Max+Plus II, Leonardo was not chosen. Leonardo has the capability to target an entered or imported design either as ASIC or FPGA. It comes with a couple of wizards to optimize, retarget and improve the design. Spectrum and Altera offer the possibility to create a working environment between MAX+PLUS II and Leonardo, which is illustrated in Table 6.1 and Figure 6.3.

MAX+PLUS II/Mentor Graphics Software Requirements

The following products are used to generate, process, synthesize, and verify a project with the MAX+PLUS II software and Mentor Graphics software:

Mentor Graphics		**Exemplar**	**Altera**
System_1076 Compiler QuickSim II Design Architect ENRead ENWrite GEN_LIB library	QuickHDL QuickHDL Pro QuickPath LS_LIB library (optional) DVE	Galileo Extreme V4.1.1 Leonardo version 4.1.3	MAX+PLUS II version 9.2

Table 6.1: Workspace Between Max+Plus II and Leonardo

The MAX+PLUS II read.me file provides more information, which versions of Mentor Graphics applications are supported by the current version of MAX+PLUS II. It also provides information on installation and operating requirements that are not mentioned in this report.

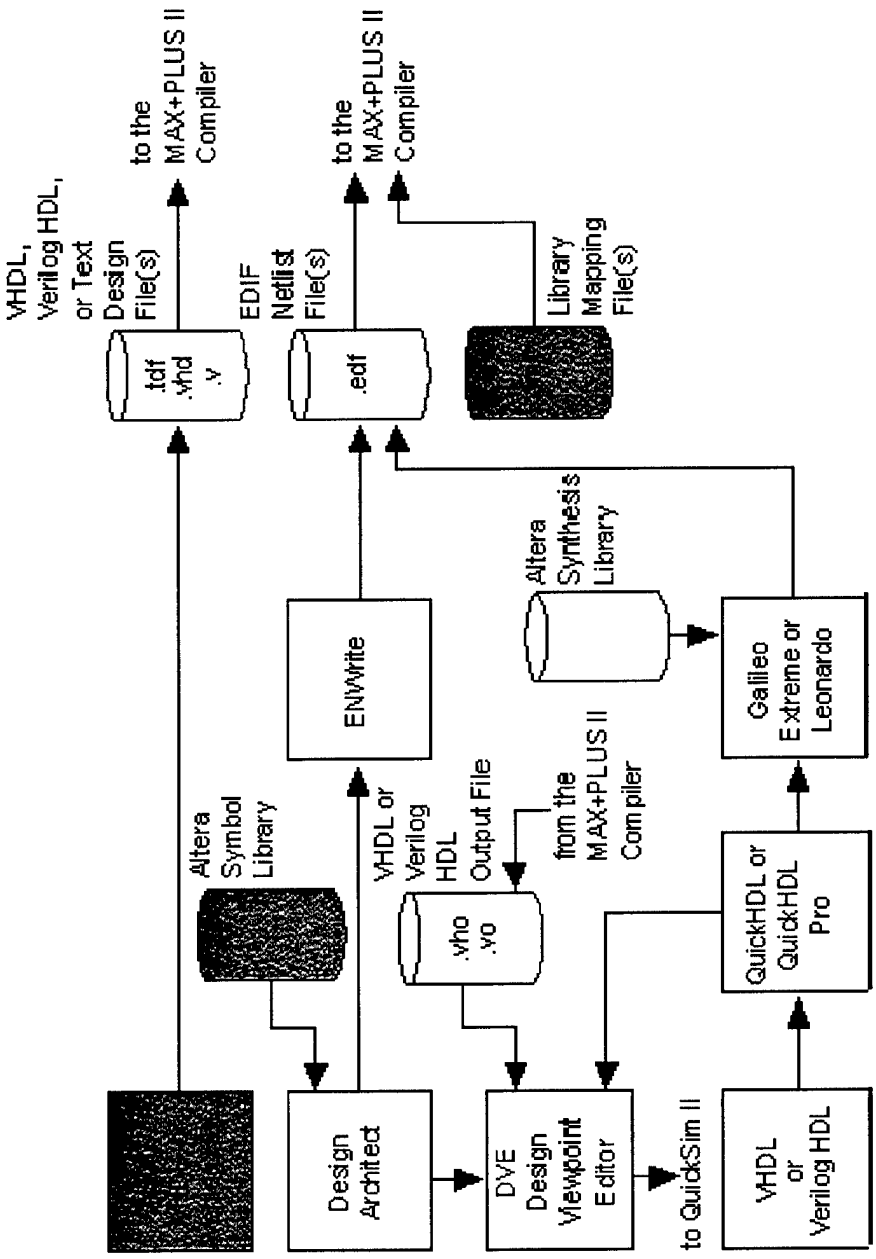


Figure 6.3: Block diagram for file flow and the MAX+PLUS II / Leonardo environment (from [9]).

The flexibility of programs like MAX+PLUS II, LEONARDO, STATECAD, etc. is determined by their ability to import files of different types. The most common types are EDIF, Verilog, and VDHL files. One has to strictly differentiate between input and output files. Output files from

Altera's MAX+PLUS II software are not compatible to input files of the same file extension, so there is a need to examine the differences in more detail. The following example is based on the MAX+PLUS II software, but is transferable to the other above mentioned programs: the input file types are VHDL, Verilog, AHDL, GDF, SCH (schematic files from ORCAD) and the EDIF files from third party synthesis tools. The output files are the files produced by Max+Plus II like VO (Verilog output netlist file), VHO (VHDL output netlist file), TDO (AHDL output netlist file) EDO (EDIF output netlist file).

The only VHDL, Verilog or EDIF files that can be generated by the Max+Plus II compiler after synthesis are post place & route netlist files. These files are normally used as an input to third party simulation tools like Verilog-XL from Cadence, Modelsim from Modeltech etc. and as input for static timing analysis tools like Primetime & Motive from Synopsys. These netlists contain the gate level description of the design and the timing delays. The EDIF input file into MAX+PLUS II is a synthesized netlist file. Therefore it is not possible to extract an input file from an output file since the output files are place & route netlists.

D. American Microsystems Inc.

Another alternative to building and ASIC from an FPGA was to contract with a company that specializes in FPGA-to-ASIC conversions, such as American Microsystems Inc. (AMI). For this approach, the entire design had to be done in FPGA oriented software like MaxPlus+ II and sent to AMI for the conversion. AMI also provides customers with their software in a light version, mentioning simultaneously, that they can't recommend this way, since the tools are very complicated and require a lot of experience [14].

This approach was not selected and not further investigated because of several reasons. First, the design conversion process yields an ASIC design that is readable by a computer and cannot be read, manipulated, and modified easily by a human, even with the appropriate CAD tools. Therefore, when the initial DIS design is eventually expanded to include more taps, the expanded design would have to be accomplished using the FPGA tools and then another design conversion would have to be performed by the contractor and paid for. Another issue is the efficiency of the design conversion with respect to speed, layout area, and power consumption of the final IC. Although great strides have been made in automated optimization for design conversion, much work still needs to be done in this area and chip designs that start life as an ASIC design usually wind up being faster, smaller, and consuming less power. Finally, one

of the goals at the NPS *Center for Joint Services Electronic Warfare* is to offer students the chance for research and the creation of projects while working towards a Master's degree. Hiring an outside firm to perform the design conversion would eliminate this opportunity in addition to being costly and not ending up with an optimum design.

E. Migration to Tanner

All the above-described processes were investigated to convert the existing FPGA design into an ASIC in order to achieve two goals. First and foremost, the high-level DIS architecture had to be fast, both with respect to high throughput and short latency. Second, the design had to be extensible, allowing an inexpensive prototype with fewer taps to be easily turned into a more finished product by just increasing the number of taps. After analyzing the original architecture, it was realized that there were several limitations being imposed on the design solely, because the implementation was done using FPGAs.

The major concern is the speed limitation, in view of the fact that the clock speed for the design should be close to 2ns. Although it might eventually be possible to meet this target with an FPGA design, in the foreseeable future, a full-custom IC will have a higher probability of meeting this speed goal.

A second major contributing factor is the physical size of the implementation. The initial, proof-of-concept design did not require a large number of taps. However, if more taps were desired to build a full operational prototype, the taps would not fit into a single FPGA. Extending the FPGA implementation to more taps would require a significant increase in the number of FPGAs. This was considered a major drawback of the FPGA implementation. Furthermore additional taps could not just be added on because the adder tree used to sum the outputs of the taps for the final output would have to be redesigned. Beyond this, as the number of taps increases, either the clock speed must be slowed down (reduced throughput) or the number of pipeline stages must be increased (increase in the total latency) to accommodate the extra delay in the additional adders in the adder tree. The total latency is the sum of the latency in the tap and the latency in the adder tree, which increases as the number of taps increases.

After considering the various different alternatives for design conversion, it was realized that a dedicated ASIC design using the Tanner Tools would be the most efficient approach. The original architecture and FPGA design allows, however, an in depth analysis of the behavior of the algorithms being implemented in the ASIC and also allows the investigation of future design concepts (for example, to counter stepped frequency waveforms).

7. Application Specific Integrated Circuit Design

A. Introduction to Tanner Tools

The Tanner Tools consist of five major integrated modules; S-Edit, T-Spice, W-Edit, L-Edit, and Nettran. The following list presents a short overview of the complete Tanner environment [15]:

Simulation Tools:

- T-Spice – an analog/digital circuit simulator
- GateSim – a gate-level simulator
- W-Edit – a waveform viewer
- L-Edit/Therm – a 3-D finite-element thermal analyzer

Front End and Netlist Tools:

- S-Edit – a schematic editor
- LVS – a layout-versus-schematic netlist comparator

Mask-Level-Tools:

- L-Edit – a layout editor
- L-Edit/SPR – an automatic standard cell placement and routing package
- L-Edit/Extract – a layout extractor
- L-Edit/DRC – a design rule checker

The ordered Tanner Tool package consists of:

- L-Edit with Design Rule Checker (DRC), Extract, and Std Place and Route (SPR)
- S-Edit (Schematic Editor)
- LVS (Layout vs. Schematic)
- T-Spice Pro with Adv Model Library
- W-Edit (Waveform Viewer)
- Tanner Tools Pro Manuals

Figure 7.1 [15] illustrates a schematic overview of the Tanner environment and the data flow between the different programs of the package. The main environment consists of the programs S-Edit, LVS and L-Edit, where L-Edit finally saves the layout in a GDSII or CIF file that is sent to MOSIS for chip fabrication. The other components may not be used but are shown for completeness.

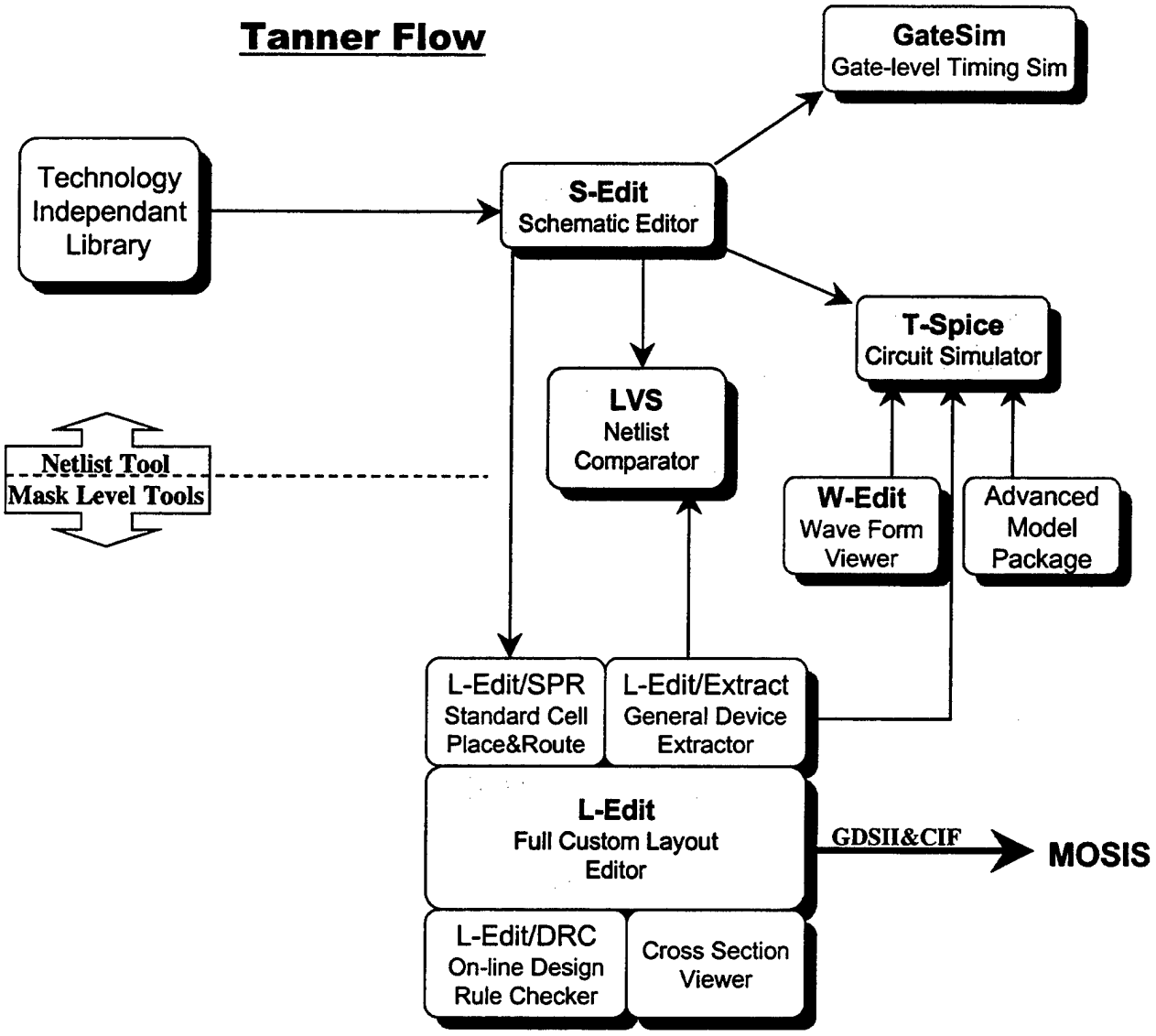


Figure 7.1: Tanner Tools block diagram (from [15]).

Nettran: Nettran is a subprogram within the Tanner Tools that has routines and libraries to import different file types. It is used as a netlist translation application to ensure file exchange between the different tools and other applications. Figure 7.2 [ ] below illustrates how Nettran fits into the Tanner Tools environment. The use of Nettran is required to translate S-Edit files into the appropriate format.

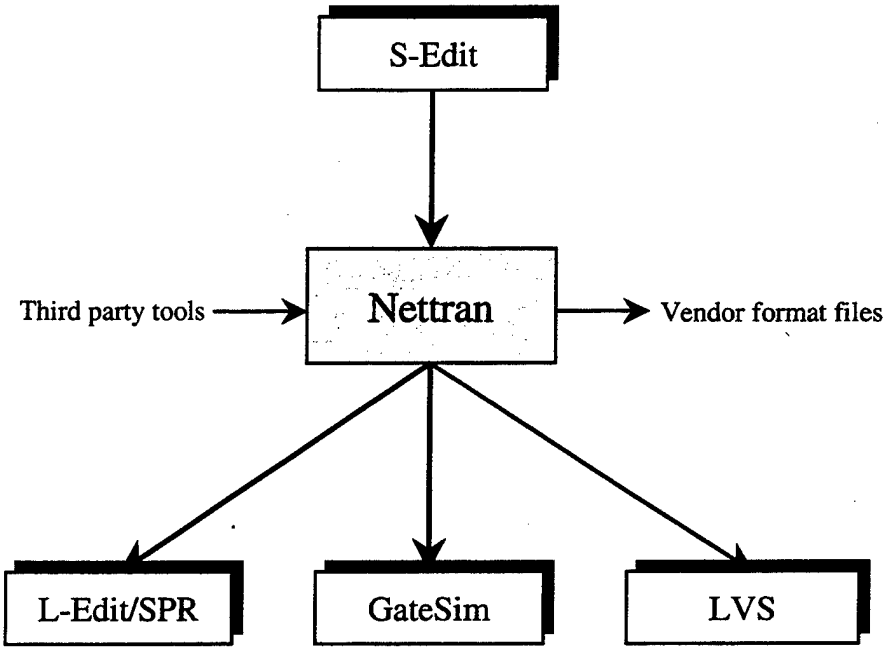


Figure 7.2: Nettran function block diagram (from [15]).

Nettran offers the capability to translate either wirelist, netlist, or EDIF files from third party tools like OrCad to standard Spice format, GateSim, or L-Edit netlist formats. Due to extended capabilities, S-Edit is now able to export a Tanner Data Base file (.TDB) that can be directly imported into L-Edit.

L-Edit: L-Edit is a physical design layout editor that creates the device level fabrication files necessary to realize the integrated circuit. **L-Edit/DRC** performs a design rule check for the intended fabrication process and can optimize the place & route. **L-Edit/SPR** generates layouts for standard cell design and can automatically construct entire chips. It includes cell placement and routing, pad frame generation, and pad routing. SPR reads netlist files produced by S-Edit and creates masks ready for fabrication. **L-Edit/Extract** creates SPICE-compatible circuit netlists

from L-Edit layouts. The output can be exported in either GDSII or CIF file format for fabrication.

S-Edit: S-Edit is a schematic editor that allows the user to enter the electronic layout of a circuit. For research and prototype devices intended for MOSIS fabrication, it contains a complete MOSIS library of components for each of the different scheduled runs. S-Edit can directly generate netlists that are usable in T-Spice simulations, where a direct link writes a complete schematic directly into T-Spice. This program is used to create the Digital Image Synthesizer circuit in order to create the new ASIC design.

S-Edit has two main workspaces, the schematic editor and the symbol editor. Besides the creation of electronic circuits in the schematic editor, the user can create a symbol for a circuit of any size in the symbol editor. Due to this possibility, S-Edit is able to handle different levels of a project. For the DIS project, S-Edit is used to construct a hierarchy consisting of five levels, where the lower levels provide the higher levels with building blocks to create more complex circuits.

LVS: LVS is a layout-versus-schematic netlist comparator to compare the exported netlist from S-Edit and the extracted netlist from L-Edit/Extract. It can also compare the layout with any other SPICE compatible netlist and ensures that both netlists represent the same circuit.

After completion of the testing phase, the layout mask will be generated in L-Edit/SPR. In order to compare the layout mask with the schematic circuit of the design, LVS is used to compare the netlists of both representations. This guarantees the equality of the layout with the tested circuit before the design is sent to fabrication. As shown in Figure 7.3, the differences after netlist comparison flow back into the T-Spice simulator and L-Edit for editing the compared files by hand. Finally this procedure will ensure the equality of the circuits.

T-Spice Pro: T-Spice Pro is a complete circuit design and analysis system which includes T-Spice, the Advanced Model Package, W-Edit, and S-Edit. T-Spice is a circuit simulator that simulates an entire circuit design with more than 300,000 circuit elements. It also has features that allow not only circuit simulation but also circuit design. The input language is SPICE. T-Spice Advanced Model Package consist of the latest transmission and semiconductor device models to achieve more realistic simulation results that are closer to real world behavior.

S-Edit (described above) provides a direct link to T-Spice, which makes the translation of the schematic design into a spice file easy. By adding parameters for testing purposes and bit pattern test vectors, the circuit logic can be tested before layout. T-Spice offers only a semi-usable algorithm for binary testing. The input data to the circuit are binary (0=0V, 1=5V), but the output will be in real voltages instead of binary word. Therefore the outputs are limited in usage. The user can however, easily develop a hard limiting function in Matlab in order to convert the output into binary values.

W-Edit: W-Edit is a waveform editor that acts primarily as a back-end data processor for the data generated in T-Spice. It is designed to display T-Spice simulation output waveforms. W-Edit was used to verify the functionality of small circuits like a register cell or a 2 input NAND gate. It is not useful for larger circuits.

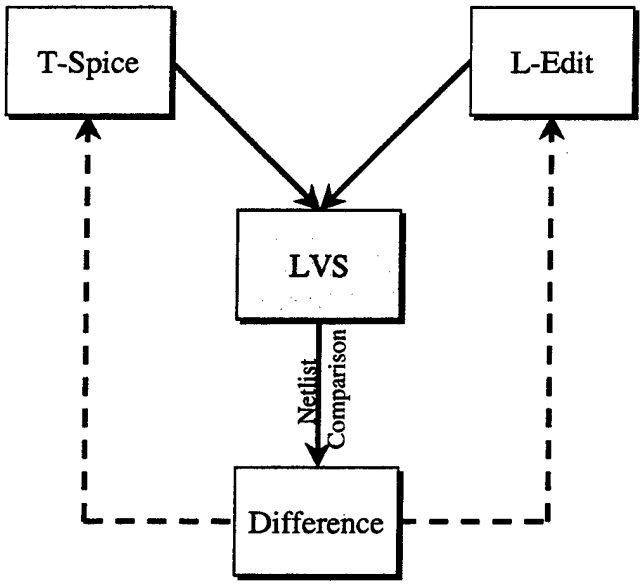


Figure 7.3: Flow for Netlist comparison.

B. Digital Image Synthesizer Architecture

This section will focus on the new DIS design and discuss its implementation in detail. The ASIC architecture is based on the modified architecture concept, where a tap and its associated range bin processing is now called a *tapline* to distinguish between the two implementations. The general data flow within a tapline is shown in Figure 7.4. The main differences between the original architecture and the modified architecture as implemented in a ASIC can be summarized as follows:

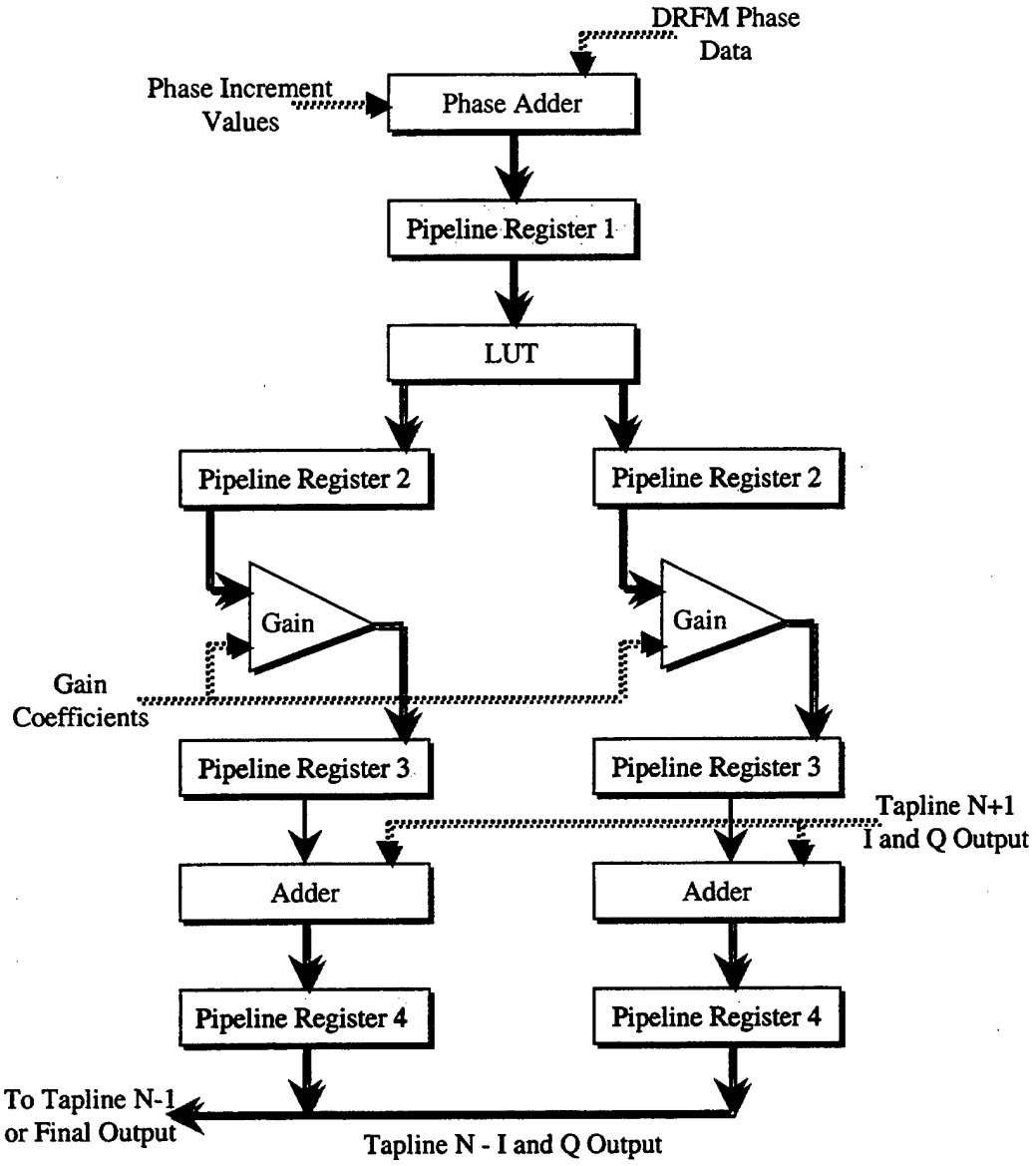


Figure 7.4: Tapline in ASIC architecture.

1. Parallel DRFM phase data input into all (32) taplines simultaneously instead of serial inputs through a tap delay line.
2. Addition of registers in the data flow of a tapline (pipelining).
3. Serial summation of the tapline data output in order to implement the necessary delay and add the output data in correct sequence for the final output.
4. Adding of a *scan path test* capability.

The DRFM phase data are the main inputs for a tapline. For now, the integrated circuit contains 32 taplines, which synchronously receive (no delay) the same clocked DRFM phase data as input. In the Phase Adder, the DRFM phase data and a phase increment value are added. The phase increment rotates for every radar pulse as already explained in the FPGA architecture discussion. The result of the phase addition in the first adder continues to propagate into Pipeline Register 1, where it is available for the LUT (Look Up Table) after the first clock cycle. The LUT uses this input as a pointer to an address space in the LUT-ROM and puts the resulting I and Q values, into Pipeline Register two. After the second clock cycle the values can penetrate into the gain block where the appropriate gain is applied. After the third clock cycle the values can enter into the second adder. The operation of this adder is to combine the inputs from

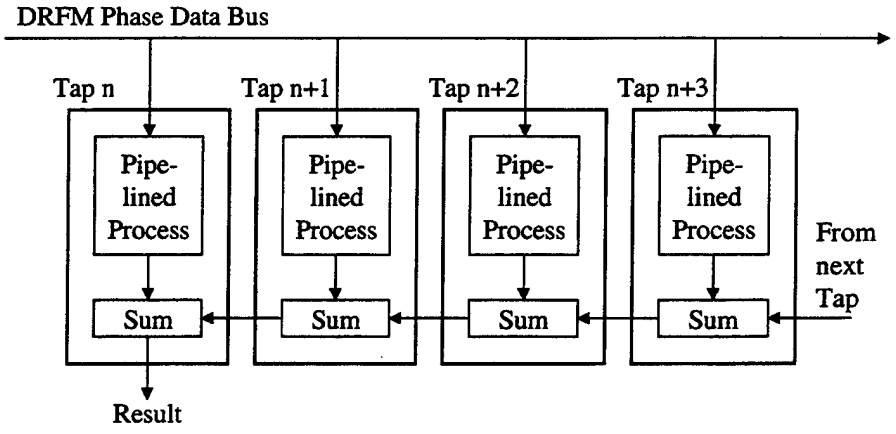


Figure 7.5: Simplified data flow in the ASIC architecture.

the other taps. Figure 7.5 illustrates the concept of the summation. To compensate for not having delay in the input DRFM phase data, delay is automatically achieved during the second addition ("Sum" in Figure 7.5) by using a pipelined adder chain instead of an adder tree. Recall that the processed DFRM phase data results in an tapline output are of the form $P_n(D_3) + P_{n+1}(D_2) + P_{n+2}(D_1)$, achieved in the original architecture by delaying the DRFM input data, propagating

through all available taps and parallel summation at the end of the process. In the modified arcitecture, the adder chain not only creates the required delay but it also eliminates the two most significant problems of the original design. First, adder chains are easily extensible since additional adders can be chained together, output to input (as long as the adders do not overflow). If the adder overflows, it is a simple process to increase the number of bits in an adder cell in a VLSI library. Second, in a pipelined design, the total pipeline latency from the first input data to the first output data is the pipeline latency in the tapline plus the pipeline delay of only one adder. Thus, as the number of taplines increases, the latency stays the same. Of course, the latency from the last phase data input sample to the last output result does increase but this is inherent in the algorithm being used and occurs in both designs.

As shown in Figure 7.4 and Figure 7.5, every tapline adds the processed data from its own line with the output data from the next higher tapline. The result is a chain of data between the first and the last used tapline. Table 7.1 illustrates this concept. The clock cycles used so far to describe the data flow in the tapline ignore the time necessary to load the inputs into the IC. The considerartion in Table 7.1 simplifies this even more and assumes, that no time is needed to process the data within a tapline. In this example tapline n (T_n) is the first one in a row of three taplines. The output of T_n is the final output, consisting of I and Q values. After clock one, every tapline produces an output with the same DRFM phase data (D_1) presented as input. With clock two the output of T_{n+2} gets added to the processed data (D_2) in T_{n+1} and the output of T_{n+1} gets added to the processed data (D_2) in T_n. In continuation of this concept, the final output is the same as for the already proven FPGA architecture.

Radar Pulse	*DRFM Data*	*CLK*	*tapline n*	*tapline n+1*	*tapline n+2*
1	D_1	0	$P_n(D_1) + 0 + 0$	$P_{n+1}(D_1) + 0$	$P_{n+2}(D_1)$
1	D_2	1	$P_n(D_2) + P_{n+1}(D_1) + 0$	$P_{n+1}(D_2) + P_{n+2}(D_1)$	$P_{n+2}(D_2)$
1	D_3	2	$P_n(D_3) + P_{n+1}(D_2) + P_{n+2}(D_1)$	$P_{n+1}(D_3) + P_{n+2}(D_2)$	$P_{n+2}(D_3)$
1	D_4	3	$P_n(D_4) + P_{n+1}(D_3) + P_{n+2}(D_2)$	$P_{n+1}(D_4) + P_{n+2}(D_3)$	$P_{n+2}(D_4)$
.	.	.	.	.	.
.	.	.	.	.	.
1	D_{62}	61	$P_n(D_{62}) + P_{n+1}(D_{61}) + P_{n+2}(D_{60})$	$P_{n+1}(D_{62}) + P_{n+2}(D_{61})$	$P_{n+2}(D_{62})$
1	-	62	$0 + P_{n+1}(D_{62}) + P_{n+2}(D_{61})$	$0 + P_{n+2}(D_{62})$	0
1	-	63	$0 + 0 + P_{n+2}(D_{62})$	0 + 0	0
1	-	64	0	0 + 0	0
.	.	.	.	.	.
2	D_1	65	$P_n(D_1) + 0 + 0$	$P_{n+1}(D_1) + 0$	$P_{n+2}(D_1)$
2	D_2	66	$P_n(D_2) + P_{n+1}(D_1) + 0$	$P_{n+1}(D_2) + P_{n+2}(D_1)$	$P_{n+2}(D_2)$
2	D_3	67	$P_n(D_3) + P_{n+1}(D_2) + P_{n+2}(D_1)$	$P_{n+1}(D_3) + P_{n+2}(D_2)$	$P_{n+2}(D_3)$
2	D_4	68	$P_n(D_4) + P_{n+1}(D_3) + P_{n+2}(D_2)$	$P_{n+1}(D_4) + P_{n+2}(D_3)$	$P_{n+2}(D_4)$
.	.	.	.	.	.
.	.	.	.	.	.
64	D_{62}	4093	$P_n(D_{62}) + P_{n+1}(D_{61}) + P_{n+2}(D_{60})$	$P_{n+1}(D_{62}) + P_{n+2}(D_{61})$	$P_{n+2}(D_{62})$
64	-	4094	$0 + P_{n+1}(D_{62}) + P_{n+2}(D_{61})$	$0 + P_{n+2}(D_{62})$	0
64	-	4095	$0 + 0 + P_{n+2}(D_{62})$	0 + 0	0
64	-	4096	0	0 + 0	0

Table 7.1: Tapline Outputs with Three Taplines.

Remarks concerning Table 7.1

Notation	Description
Radar Pulse	Represents one radar pulse split into DRFM phase data
DRFM Phase Data	62 DRFM phase data samples per radar pulse
CLK	Clock pulse for the example.
Tapline n	Output of the n^{th} tap.
Tapline n+1	Output of the (n+1) tap.
Tapline n+2	Output of the (n+2) tap, last tap in the example
$P_{n+x}(D_x)$	Processed phase data sample available at designated tapline output.

Table 7.1 ignores the time that is needed to process data within a tapline. To complete this discussion, Table 7.2 summarizes the amount of time in terms of clock cycles from presenting new DRFM phase data to a tapline until the final output at the end of the tapline.

Clock Cycle	Output available at:
0	Phase Accumulator
1	Output of Pipeline Register 1 (5-bit)
2	Output of Pipeline Register 2 (8-bit)
3	Output of Pipeline Register 3 (11-bit)
4	Output of Pipeline Register 4 (16-bit), end of tapline

Table 7.2: Clock cycles in a tapline.

Before data can be processed in a tapline, the data needs to be loaded into the integrated circuit. In spite of this fact this section considers only the concept of a tapline, with the load cycles addressed later on in this section.

Due to the adaptation of registers between the building blocks of a tapline, it was possible to install a test path to improve the testability and functionality for the entire IC. This scan path test capability can be used to strobe values into the registers to produce results for special test cases. The test vectors within the registers can then be processed for a desirable number of clock cycles before they are read out again. The implemented scan path is also part of the discussion later on in this chapter.

C. S-Edit Implementation

The following section provides information how the concept is implemented into S-Edit. Since the program supports design hierarchy, the DIS was divided into five design levels. To increase the functionality and the signal flow control several control signals were introduced. These control signals are also used to indicate the states (valid/not valid) of the output data. Furthermore a scan path test capability was installed to enhance the testability for sub-levels during the test phase and to verify correct IC operation.

Design Hierarchy: S-Edit is a schematic editor to enter an electronic layout of a circuit. It is capable of creating hierarchical circuits by using predefined or user created modules. Tanner provides the customer with a great variety of modules and building blocks, but only a few of them were used to build circuits on the lower levels.

The DIS design in S-Edit consists of five levels. The first level is represented by low-level building blocks like transistors and logic gates. Using blocks from lower hierarchy levels allows creating higher levels in order to increase the complexity stepwise. This concept provides two main advantages:

1. The logic of the design is more obvious, easier to understand, and easier to verify.
2. The layout editor (L-Edit) can use the same hierarchy to synthesize the mask stepwise. Since the hierarchical layout process allows a slow increase in the complexity, the layout editing is easier, more reliable, more efficient and faster.

The following hierarchy tree illustrates the structure of the DIS in S-Edit:

Level 1: elementary building elements from existing libraries or modified library elements. This includes for example, a register cell, an adder cell, a Mux2, transistors and all logic gates.

Level 2: builds on elements from level 1 to create: 5-to-32-bit decoder part 1, 5-to-32-bit decoder part 2, the LUT-ROM, Gain Shift, N-bit register, and N-bit adder.

Level 3: makes use of the elements from level 2 and 1 to build the tapline.

Level 4: the Supertap and the Supertap Mirror consist of level 3 and level 1 components.

Level 5: includes and 5-to32-Bit LUT and extends the concept of level 4 components to create the top level circuit with input pads and output pads.

These levels will be discussed in more detail. A complete graphical representation of the building blocks can be found in the Appendix, divided into symbols and schematics for all sub-circuits used in S-Edit.

Architecture Circuit Description in Level 1:

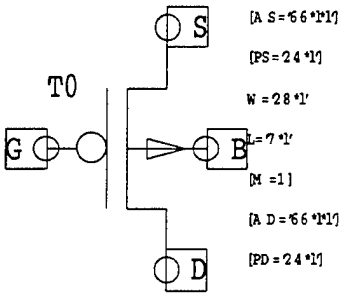


Figure 7.6: P-FET transistor.

The very basic elements in the new design are the P-FET and the N-FET transistors. These types of transistors are represented only by a schematic appearance, which is specified by a Spice output statement, as shown in Figure 7.6. The Spice output defines the transistor in gate length and width and provides even more editable parameters that are not relevant for the current design and therefore are not mentioned here. The important parameters are multiples of lambda (l), so the transistor is scaleable and can be used for different layout processes. Due to cost reasons and availability of certain process runs at MOSIS, it was decided to go with a HP 0.5µm process. Nevertheless, if the concept is proved and the IC is fully operational, the target process can be easily changed to a smaller (faster) size without any changes to the existing design in S-Edit.

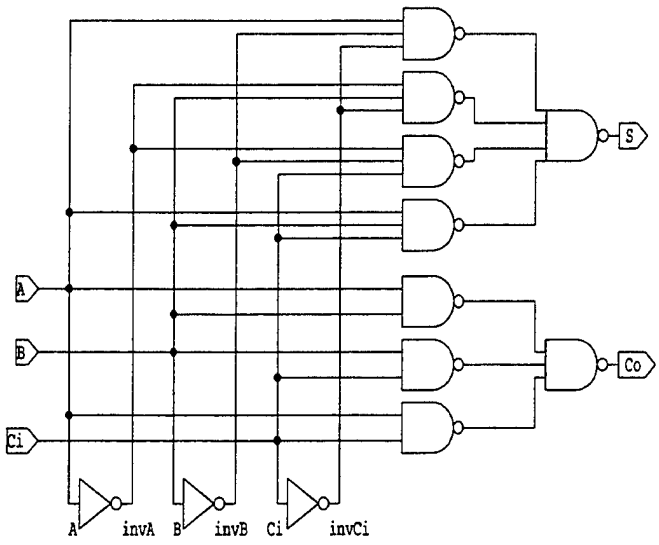


Figure 7.7: Adder cell.

The adder cell and register cells, as shown in Figures 5.7 and 5.8, are building blocks to create n-bit adders or n-

bit registers. The adder cell can add two 1-bit binary input words. The input pads are labeled A and B, where the third input pad, Ci the carry-in bit is used to connect two or more adder cells. The carry output pad, Co and the output pad S define the 2-bit output word. The function of the cell is described by the following two equations:

$$S = invB * invCi + invA * B * invCi + \\ + invA * invB * Ci + A * B * Ci \quad (7.1)$$

$$Co = A * B + B * Ci + A * Ci \quad (7.2)$$

To build an n-bit adder, Co of cell N has to be connected to Ci of cell N+1. A 5-bit adder and a 16-bit adder are part of level 2 in the hierarchy.

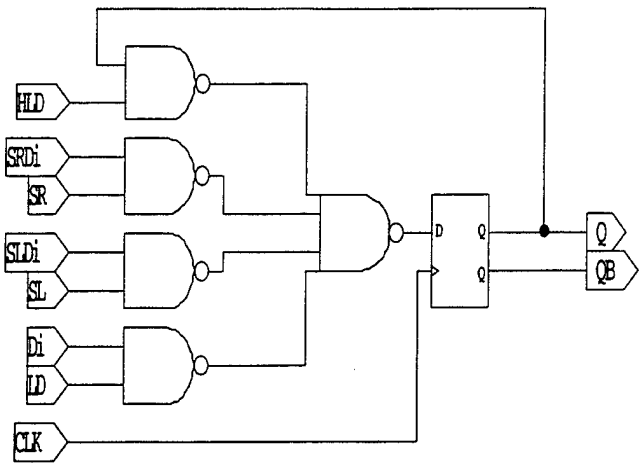


Figure 7.8: Register cell.

The register cell is used to implement the scan path test and to give control over the data flow in the tapline logic. The control logic consists of hold, load, clock, and the scan path pads. Besides the clock, only one of the control signals is allowed to become high at the same time. If load goes high, the register performs normal operations and "clocks" the input to the output. A logical high for "hold" freezes the last processed value and ignores new input data. If all control pads are low at the same time, the register is forced to perform a synchronous clear (all outputs become low). To make an n-bit register, Q of cell N must be connected to SRDi of cell N+1 and Q of cell N must be connected to SLDi of cell N-1, where the register control pads are connected in parallel. A 2-bit register, a 4-bit register, a 5-bit register, a 8-bit register, a 11-bit register, and a 16-bit register are part of level 2 of the hierarchy.

Architecture Circuit Description in Level 2:

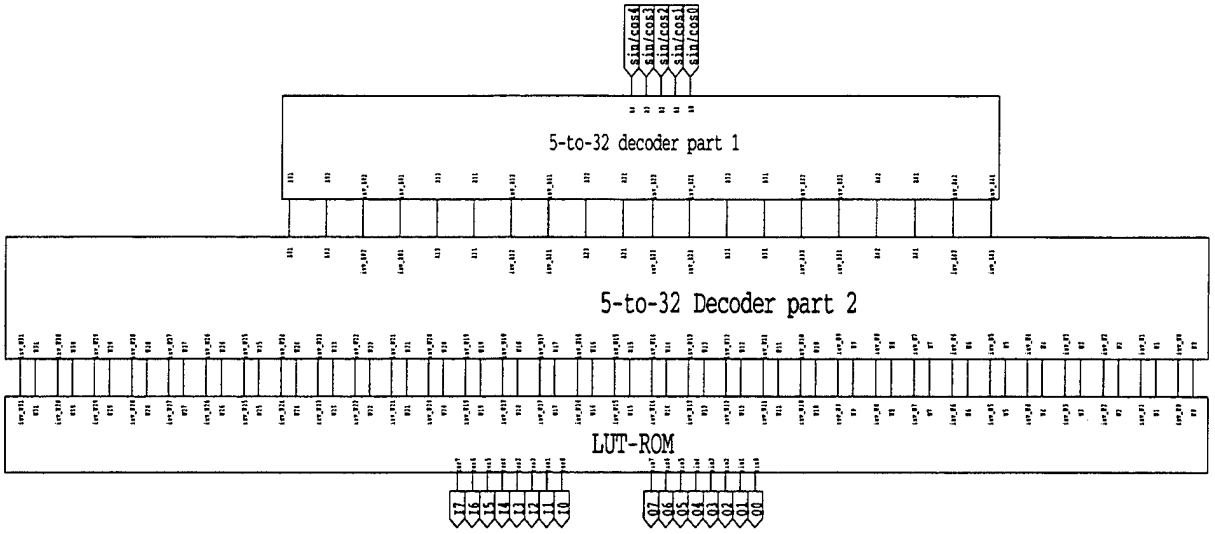


Figure 7.9: LUT module.

Level 2 elements are the building blocks for a tapline. The Look-up-Table (LUT) is shown in Figure 7.9 and is a composition of three sub building blocks that are shown in the Appendix: 5-to-32 bit decoder part 1, 5-to-32 bit decoder part 2, and the LUT-ROM. The two decoder translate the 5 bit input word into an address space of 32 possible inputs, that are used to

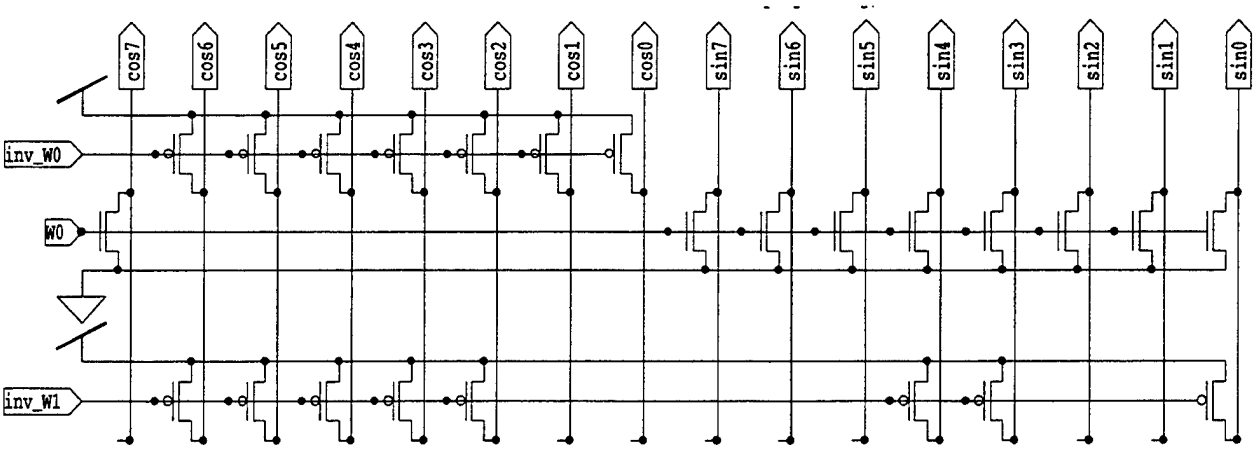


Figure 7.10: Part of LUT-ROM.

trigger the LUT-ROM. Depending on this input, the ROM module makes the corresponding table look up available at the output. Figure 7.10 shows only a small part of the LUT-ROM to illustrate

the structure. The entire block is shown in the Appendix. The ROM consists of 32 double rows of transistors with a length of 16 transistors per row, each divided into two columns. Every double row represents a 2 * 8 bit value, the cos (I) and sin (Q) outputs. The code values are created by adding P- and N-FET transistors as shown in Table 7.3 below:

Program a "0"	Program a "1"
Set N-FET transistor in row for input pad W_n	Set P-FET transistor in row for input pad inv_W_n

Table 7.3: LUT Programming.

The LUT block requires a 5 bit input word that will be translated in two pre-programmed, 8 bit output words. Note that the Altera design used two tables, one for I outputs and one for Q outputs. Due to a extended-decoder architecture in the modified design, both tables could be combined in one building block to reduce the amount of transistors.

The Gain Shifter "multiplies" the input by performing a shift of the input word. The binary input pads Gain0 and Gain1 as shown in Figure 7.12 determine the amount of shift or gain applied to the two's complement input word. Table 7.4 illustrates the gain effects on the input.

Binary Input		Gain	Multiplication	Effect on binary input word
Gain1	Gain0	Factor	Factor	
0	0	0	1	No effect on input word; input = output
0	1	1	2	Input word gets shifted by one digit to the left, For example: input = 1101 Output = 11010
1	0	2	4	Input word gets shifted by two digit to the left, For example: Input = 1101 Output = 110100
1	1	3	8	Input word gets shifted by three digit to the left, For example: Input = 1101 Output = 1101000

Table 7.4: Gain Shift.

Gain0 and Gain1 are loaded into a two-bit register, where they are available for the Gain Shifter as inputs. The gain factor in Table 7.4 is the integer representation of the two gain inputs. They are related to a multiplication factor as specified in the Matlab m-file Range-Doppler-Amplitude Map Entry Program already mentioned and described in previous chapters. The Gain Shift block is a composition of slightly modified Mux2 Tanner library elements, where one is shown in Figure 7.11. The modification was necessary, because of the reverse output available compared to the Altera Mux2 output. The logic equation of the modified Mux component can be described as:

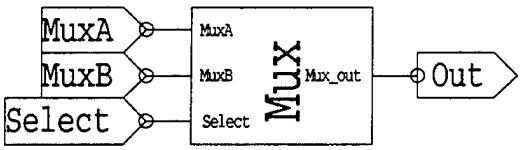


Figure 7.11: Mux symbol.

$$Out = MuxA * not\_Sel + MuxB * Sel \tag{7.3}$$

Figure 7.12 illustrates the concept of the Gain Shift block. It shows the logic that leads to the shift of the two's complement binary input. The Gain Shift block or Gain Modulator requires an 8-bit two's complement input word and two Gain coefficients (Gain0, Gain1). Due to the highest possible shift of three digits with gain coefficients of Gain0=1 and Gain1=1, the output word can be an 11-bit two's complement binary number.

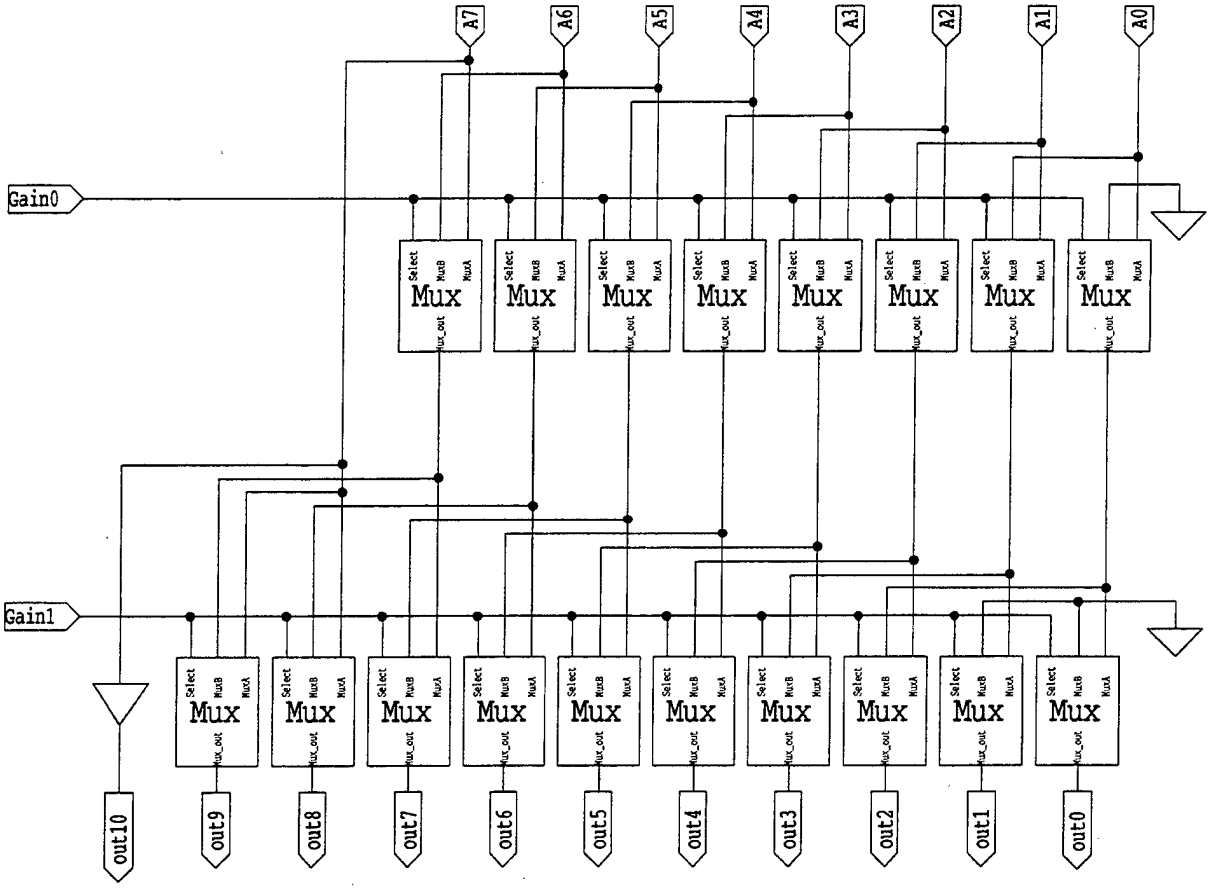


Figure 7.12: Gain shift block.

Architecture Circuit Description in Level 3:

The tapline is shown in Figure 7.13 and is the only element in level 3. It is considered to be the central building block of the DIS since every other block in higher design levels is an extension of this block. The chip capabilities are directly related to the number of taplines implemented in the chip. Every tapline that is added to the chip extends the possible size of the false target. In reference to the Range Doppler Map Entry, one tapline in the hardware

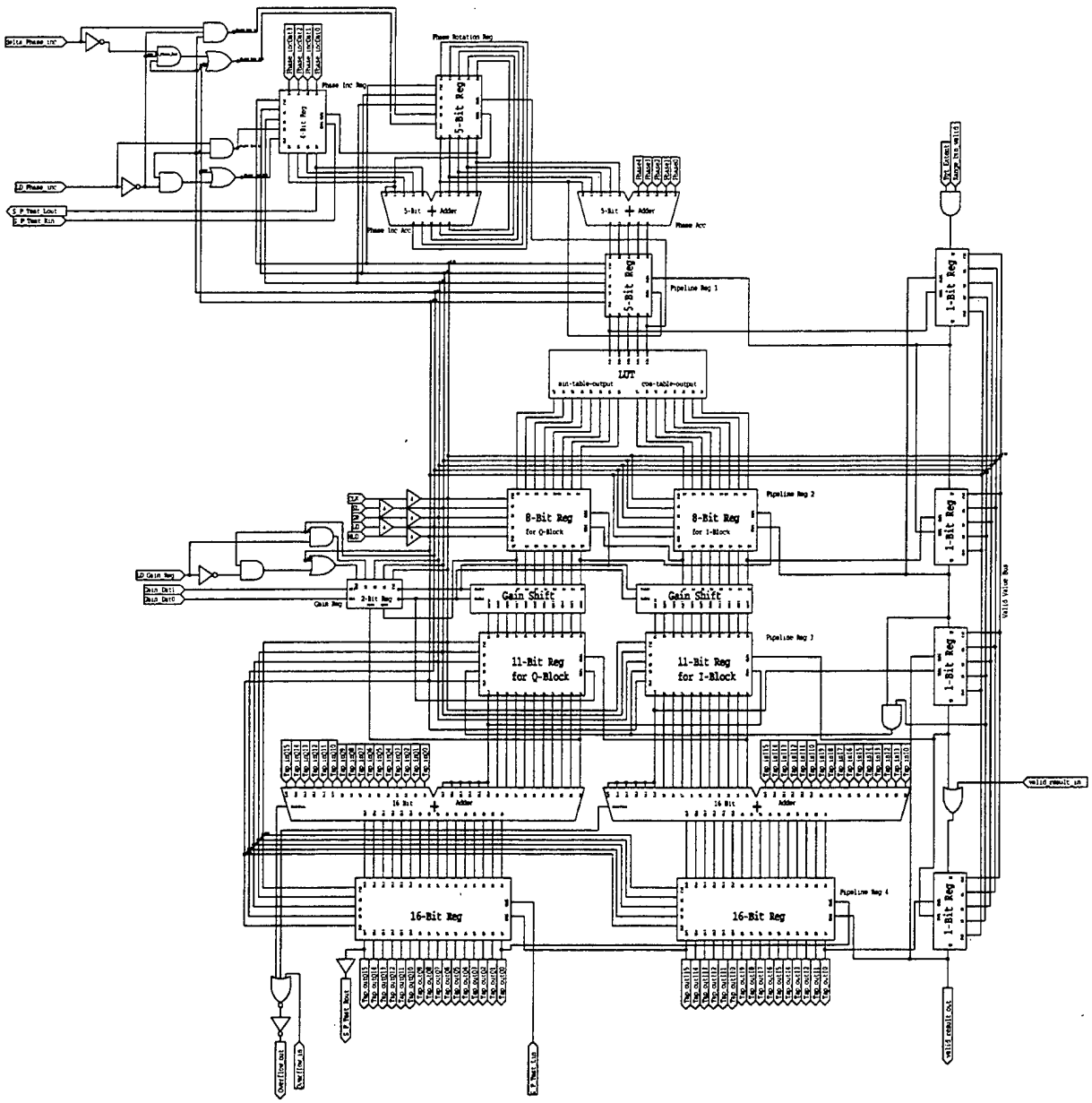


Figure 7.13: The tapline.

represents a single cell in the Range-Doppler Map. Currently the chip design contains 32 taplines. Therefore the target extent is 32 cells in the "Range-Doppler-Amplitude Map Entry program", which can be related to a physical extend of the false target of

$$1.2\text{m (for each cell)} * 32 \text{ taplines} = 38.4\text{m}. \qquad (7.4)$$

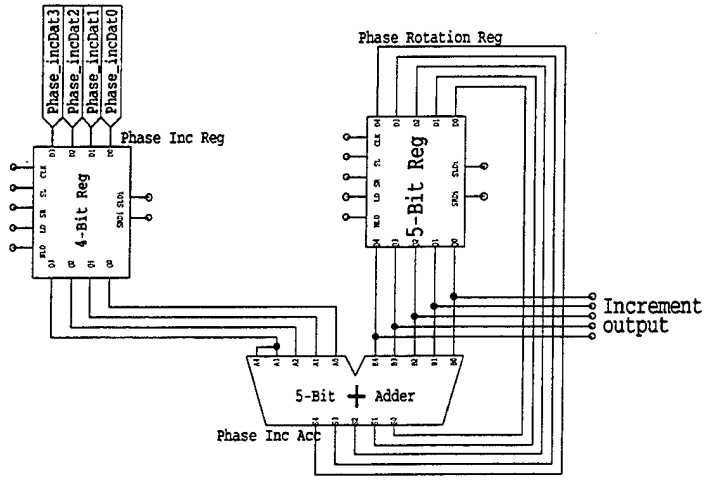


Figure 7.14: Tapline phase increment.

Due to the complexity of the tapline, it is split into three sub blocks. The control signals and the scan path test are not part of the discussion and are excluded for now. The first block is called the Phase Incrementer and is shown in Figure 7.14. It consists of a 4-bit register, a 5 bit-register and a 5-bit adder and requires a 4-bit binary input word. The two's-complement binary input is the desired phase increment value ωPRI that has to be added to the DRFM phase data. The phase incrementer supplies integer multiples of the desired phase increment ($n\omega PRI$) to the phase data on a pulse-to-pulse basis. That is, due to the phase rotation requirement, the output of the phase incrementer has to change (increase or decrease) by the amount of the phase increment value for every new pulse. The increment value supplied to the tapline may be constant over several radar pulses (constant Doppler frequency). To achieve this "constant" adding the increment value is used as the first input for the 5-bit adder, where the connection between bit 4 and 5 is the sign extension. The adder output goes into a 5-bit register that is connected to the 5-bit adder. Due to this construction, the output of the 5-bit adder is always a n-multiple (n = 1, 2, ...) of the original input. The phase rotation can be adjusted by control signals, which control the registers in this block. The master clock controls the overall behavior of the registers. Since a register needs one clock cycle to produce a valid output, the Phase Incrementer has a requirement of at least one clock cycle before a valid result is at the output. The phase increment in the 5-bit register needs to be activated exactly one clock cycle before a new pulse with new DRFM phase data can be

processed. The output of the Phase Incrementer is a 2-complement binary word, which is the input to the next logical block, the LUT-Module.

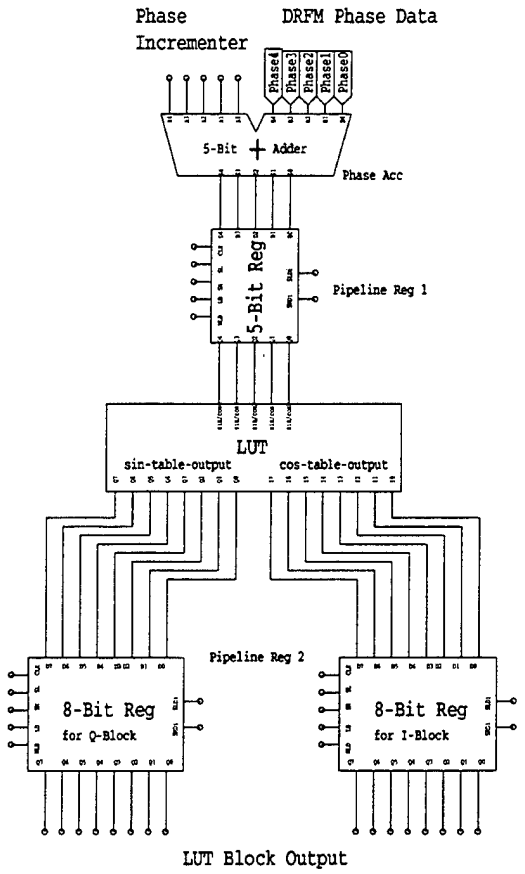


Figure 7.15: Tapline LUT-module.

The LUT-Module shown in Figure 7.15 is the second sub block of the tapline. It consists of a 5-bit adder, a 5-bit register, a LUT for I and Q values, and a 8-bit register. The adder takes the DRFM phase data inputs and the Phase Incrementer outputs and adds them together. Note that the addition of two 5-bit binary words could result in a 6-bit word. This fact can be ignored, since the Phase Incrementer output is a phase value repeating over a period of 2pi. Therefore the adder output, in conjunction with the cos and sin LUT, can also be seen as a periodic output over 5-bit.

The adder output goes into a 5-bit register (Pipeline Reg 1), where it is available at the output after one clock cycle. The LUT block takes the 5-bit input word and uses it as an address to find the corresponding I and Q values in the sine/cosine LUT as described earlier. From this point the signal flow is divided into two streams (I and Q values). Nevertheless the operations performed on these values will be the same up to the final output at the end of the first tapline. The LUT output data are inputs to an 8-bit register (Pipeline Reg 2) and become available at the output after the next clock cycle.

The third sub block of the tapline is the "Gain and Adder block" as shown in Figure 7.16. It consists of a Gain Shift block with a gain register, a 16-bit adder, and a 16-bit register each for the I and Q part of the tapline (compare also with Figure 7.13). The input to the Gain Shift is the output of the LUT. As described earlier, the Gain Shift performs a shift of the input values in accordance with the specified gain coefficients. The output results in a 2-complement 11-bit word for each stream. After one clock cycle the values are present at the output of the following 11-bit register (Pipeline Reg 3).

The 16-bit adder takes two inputs. One is the output of the Gain Shift block, which again requires a sign extension to the most significant bit achieved by the connection of bit 11 to 15. The second input is the output of the tapline that is the next higher in a line of 32 taplines. This illustrates the concept mentioned at the beginning of this chapter. To recall, imagine that the so far considered tapline is tapline #1. The next higher tapline is #2. After 4 clock cycles the first outputs at both taplines are available (see Table 7.2), where tapline #2 presents its values to the 16-bit adder of tapline#1. With the next clock cycle, these output values of tapline #2 present at the adder get added to the next data in tapline #1 coming out of the 11-bit register after the Gain Shift. After the addition in the 16-bit adder the values are present to the input to the last register (Pipeline Reg 4) in the tapline. After one more clock cycle, the values in form of 16-bit 2-complement words are available at the tapline output.

All of the three sub blocks that were discussed have control and test signals managing the data flow to achieve the physical requirements for the DIS. These signals are not part of this section but are discussed in detail later on. Furthermore, there are four 1-bit registers on the right hand side only shown in Figure 7.13, that were not part of the discussion either. These registers are used to allow particular control signals to penetrate through the tapline synchronously with the clock.

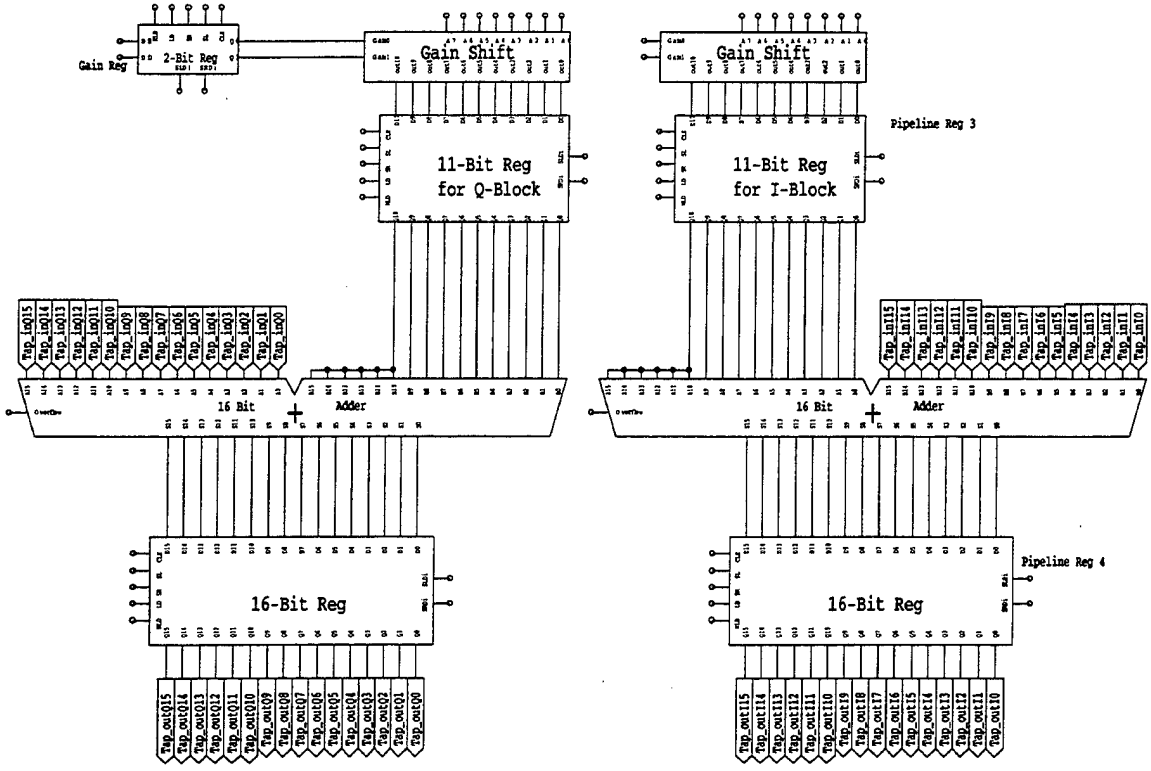


Figure 7.16: Tapline gain and adder block.

Architecture Circuit Description in Level 4:

Level 4 of the design construct in S-Edit pursues the tapline regarding the addition. Figure 7.17 shows a so-called Supertap, which consists of eight taplines connected in series. A Supertap has the same control signals and output pads as a tapline, but requires more input pads for signals that are not the same for every tapline. For example the gain coefficients, the phase increment values, and the target extent control signal are different for the distinguished taplines. The target extend is explained under the control signal section and will be disregard for now. The gain coefficients and phase increment values are important values for the false target generation.

Due to the addition process in the 16-bit adder in the new architecture the DRFM phase samples are the same for every tapline as illustrate earlier in this chapter. The 2*16-bit

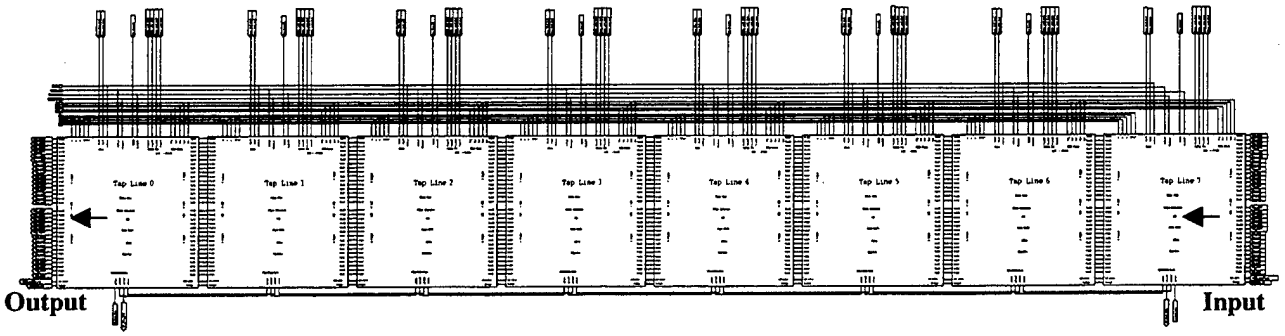


Figure 7.17: Supertap schematics

input pads on the right hand side of Figure 7.17 can be used to connect another Supertap in series. Due to this open concept of serialising Supertaps, in theory any number of Supertaps could be easily chained together without a need to modify the existing design. The number of Supertaps is therefore limited only to the size of the chip and the current available technology for mask layout. If the desired false-target-extent is larger than the available Supertaps that could fit into an IC, several ICs can be "daisy-chained" together to increase the possibilities for false target generation.

Architecture Circuit Description in Level 5:

Level 5 is the highest level of the current design. As shown in Figure 7.18, four Supertaps are connected in series to get an overall number of 32 taplines (eight taplines per Supertap) for the first IC production. A 5-to-32-Decoder is used to control the target-extent control signal in the form of a truth table. If the required target size is smaller than the available number of taplines, it is not necessary to use all taplines. The decoder generates control signals to activate the needed number of taplines. To illustrate, a five-bit input word can represent numbers between 0 and 31. These numbers are directly related to the tapline enumeration as shown in Figure 7.17. If the generation of a false target requires 5 taplines, the binary input to the decoder is 00110, which activates tapline 0 to 4 in Supertap A (lower left corner in Figure 7.18). The schematics of the decoder can be found in the Appendix.

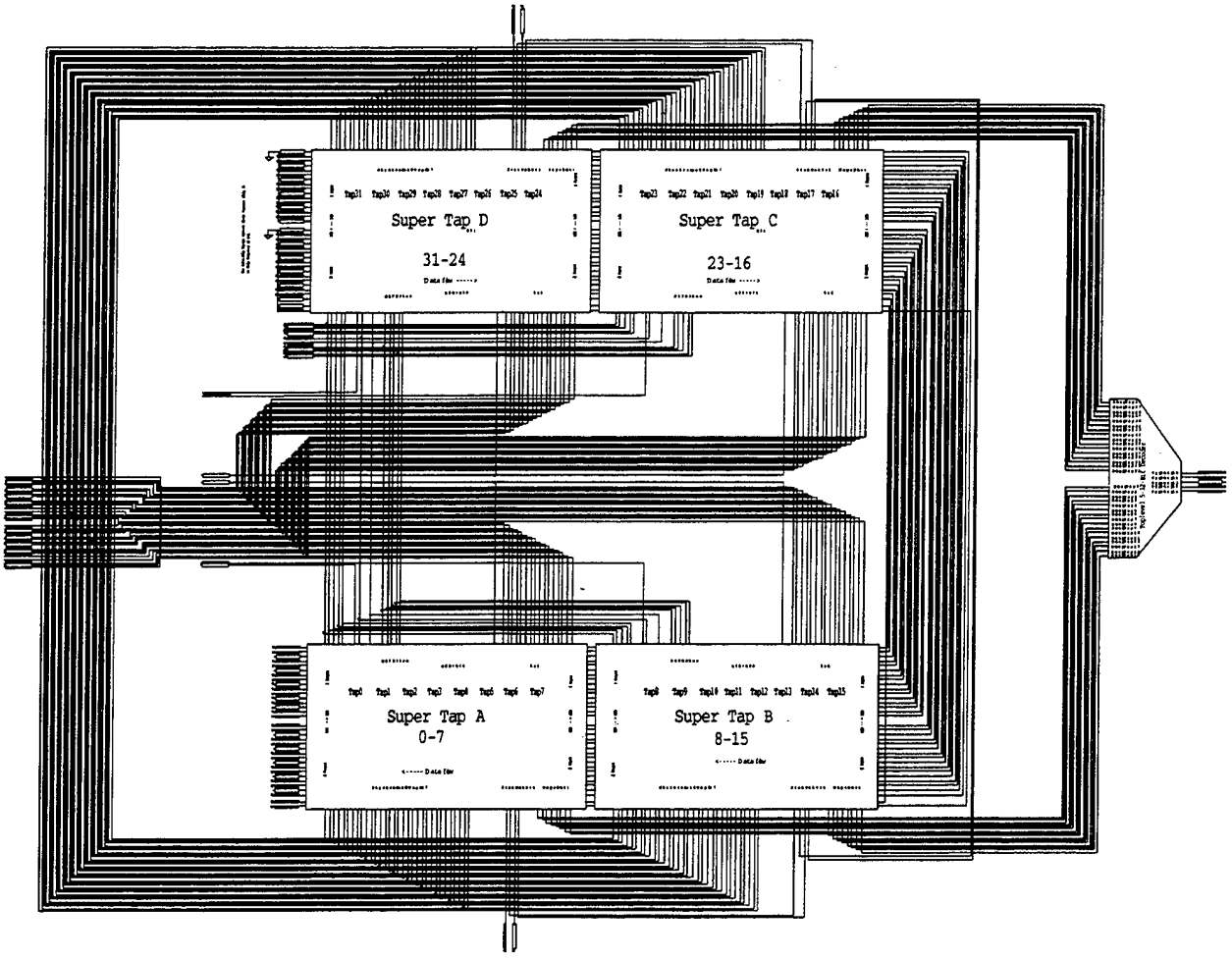


Figure 7.18: Toplevel of the DIS ASIC architecture

The input and output pads used in Level 5 can be described as follows:

Inputs:

- Supertap D has two16-bit input pads for I and Q values to connect Supertaps in series. For this layout, they are connected to ground to exclude possible side effects based on pending or floating nodes.
- 5-bit binary word for the 5-to-32-Decoder to control the target, as mentioned above.
- A master clock, which is connected in parallel to clock input pads throughout all five levels.
- Control signals for the scan path test (SR, SL, S_P_Test_Rin, S_P_Test_Lin)
- 2 * 32 = 64 gain coefficients (Gain0/Gain1 for each tapline)
- 4 * 32 = 128 phase increment values (phase_inc0/1/2/3 for each tapline)
- 5-bit DRFM signal (same input for all tapline)
- 1-bit control pads to control the data flow in the chip (load_phase, delta_Phase_increment, Range_bin_valid, Load_Gain_Reg and overflow_in). These signals are discussed in the next section.

Outputs:

- Final summation in Supertap A (output of I and Q values for the false target), which can be used to import the results into Matlab for plotting.
- Control signals for the scan path test (S_P_Test_Rout, S_P_Test_Lout)
- 1-bit control pads to verify the output result of the chip (OVERFLOW_OUT AND VALID_RESULT_OUT). These signals are discussed in the next section.

With a total of two gain coefficients and four phase increment inputs per tapline and 32 taplines for the current design, a total of 192 input pins are required. Adding this many pins to the number of high-speed input and output pins would greatly increase the cost of IC fabrication and the complexity and cost of using the finished IC in a system. Furthermore, when the number of taplines is increased in the future, this problem would become even worse. However, the inputs to set the gain coefficients and the phase increment value should only change at the beginning of a new radar pulse, not on every sample within a radar pulse. Therefore, the gain coefficient and phase increment inputs are relatively low bandwidth and can be bussed together. To maintain compatibility with off-the-shelf, digital signal processing microprocessors and components, a 32-bit input bus has been designed for the top-level design. The 64 gain coefficient inputs for 32 taplines (2 per tap) are loaded in two bus cycles. The 128 inputs for 32

taplines (4 per tap) for the phase increment are loaded in four bus cycles. Table 7.5 lists the bus cycles and the control signals, and represents an example for how the inputs could be loaded into the IC.

Bus-CLK	Control Signal	Function
1	Load gain Reg Tap 0-15	Loads the gain coefficients for tapline 0-15
2	Load gain Reg Tap 16-31	Loads the gain coefficients for tapline 16-31
3	Load Phase Inc Supertap A	Loads the 4-bit phase increment value into taplines 0-7
4	Load Phase Inc Supertap B	Loads the 4-bit phase increment value into taplines 8-15
5	Load Phase Inc Supertap C	Loads the 4-bit phase increment value into taplines 16-23
6	Load Phase Inc Supertap D	Loads the 4-bit phase increment value into taplines 24-31

Table 7.5: Loading of Bussed Inputs

The current design can be easily expanded to include more than 32 taplines without a further increase in on-chip hardware related to the bus or the number of I/O pins. All that it necessary is to connect the gain coefficient inputs and phase increment inputs from each tapline to the bus and increase the number of bus cycles required to load the data into the chip as shown in Table 7.5.

In level 5, all inputs and outputs are attached to Pad blocks. A Pad consists of a Buf4 and a PadIn or PadOut for an input or output respectively; figure 7.19 shows an output pad. A Buf4 is a cell that does not perform any logic function but does provide buffering of logic signals. It can be driven at high speed by a minimum-sized logic gate. It is capable of sinking and sourcing 4 times the amount of current that a minimum-sized logic gate can sink or source. Therefore it is very good for driving networks that have a high fan out and large capacitive loads, such as clock and control

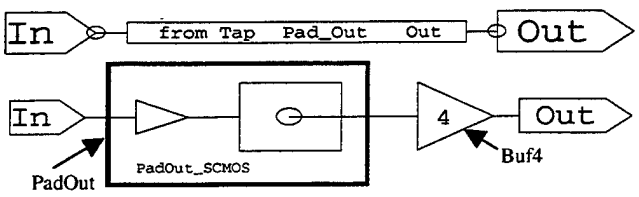


Figure 7.19: Output pad

signals and is used throughout the design. PadIn is used to connect signals from outside the IC to the on-chip inputs. It provides a bond pad site for a wire bond, a static discharge protection circuit, and logic signal buffering to drive high-fanout and high-capacitate on-chip networks. PadOut is used to connect the IC outputs to the off-chip networks. It provides a high power driver circuit, a static discharge protection circuit, and a bond pad site for the wire bond.

D. Chip Operation

This section provides an overview of the inputs that control the chip and the signal-flow operation. Also discussed in the concept of *scan path testing* illustrating the implementation into the new ASIC design.

Control Signals:

CLK (clock): CLK is the master clock for the entire chip and controls every register in the chip.

LD (load): LD is a bit to control the behavior of the registers. If LD is high, Shift Right (SR), Shift Left (SL) and Hold (HLD) have to be low. This setup is the mode for normal operation. If LD is low, the chip is in a special mode, where HLD or Scan Path Test controls can be used.

HLD (hold): If HD is high, SR, SL and LD have to be low. This bit is used to hold a value within a register that should not change over a period of clock cycles as long as HLD is high. The input values to a register will simply be ignored and the last values before HLD got high are retained. If all control signals (HLD, SR, SL, and LD) are low at the same time, the register performs a synchronous clear and the output will be entirely low.

Delta Phase Inc: Delta Phase Increment is a control bit for the Phase Rotation Register (Phase Rot Reg). Since a CHIRP pulse or radar pulse is divided into samples and the phase should only rotate once for every pulse, the Phase Rotation value has to be incremented from pulse to pulse. If Delta Phase Inc and LD are high, the phase increment can "rotate" controlled by the clock. The resulting value will

be added to the DRFM input data. If Delta Phase Inc and LD Phase Inc are low, the Phase Rot Reg is in hold modus and the phase increment value, as input for the Phase Accumulator, is fixed. Before processing a new Radar pulse, the phase has to rotate once to produce a new incremented phase value, which is the same for all samples within a pulse.

LD Phase Inc: Load Phase Increment is a control bit, that effects the two registers for the phase rotation block in order to signal a change for the phase increment value. If this bit and LD are high, a new phase value gets loaded and the Phase Rot Reg performs a synchronous clear. If LD Phase Inc is low and LD is high, the Phase Inc Reg is in hold modus in order to keep the phase increment value constant over the length of a radar pulse.

The inputs for the gain coefficients and the phase increment values are bused on a 32-bit bus. There are 4 phase increment values per tapline and 32 values per Supertap, therefore the chip needs to load 128 phase increment values to be able to process DRFM data. Since the bus has a length of 32-bits, four bus cycles are needed to load the gain coefficients, controlled by "Load Phase inc Supertap A-D".

LD Gain Reg: Load gain Register is a control bit, that effects the behavior of the gain register in order to signalize the change of gain values within a tapline. To load new gain values, LD Gain Reg and LD have to be high. If LD Gain Register is low and LD is high, the gain register is in hold modus.

The inputs for the gain coefficients and the phase increment values are bused on a 32-bit bus. There are 2 gain coefficients per tapline and 16 coefficients per Supertap. Therefore the chip needs to load 64 gain coefficients values to be able to process DRFM data. Since the bus has a length of 32-bits, two bus cycles are needed to load the gain coefficients controlled by "Load Gain Reg Tap 0-15" and "Load Gain Reg Tap 16-31".

Tgt Extent: The Target Extent control bit is used to activate or deactivate taplines, regarding of the size of the false target. The chip is able to handle a false target up to 32 cells regarding to the in MATLAB constructed range-Doppler

map. If the desired false target is smaller, less taplines are necessary to build the false target. In level 5 of the design hierarchy a 5-to32-bit decoder uses a truth table to adjust the used taplines in dependency of the target size. If a target needs for example only 12 tapline, the Tgt_Extent for the first 12 taplines are high. The Tgt_Extent for the other taplines are low and the output values are ignored. Therefore the false target output is available in less time.

Range Bin Valid: A tapline needs four clock cycles to produce a valid output. Range bin Valid gets high when new values are presented to the tapline. The bit penetrates through 1-bit register cells to the Valid Result Out pad. If Valid Result Out gets high, the output out of the tapline is fully processed and the output is valid.

As long as the 1-bit Pipline Reg 3 doesn't receive a high for Range Bin Valid, the register after the Gain Shift block gets cleared for every clock cycle. A higher tapline can produce valid results, even if a lower one cannot. Therefore the lower tapline is not allowed to add garbage data to a valid output of the higher tapline within the 16-bit adder, so that the input of the lower tapline to the adder has to be zero.

Valid Result In: Valid Result in performs a similar operation as Range Bin Valid. This input port is connected to the Valid Result Out port of the next higher tapline. If the next higher tapline produces a valid output that is lead into the lower tapline, Valid Result In is high and the next lower tapline produces a valid output with the following clock, although the lower tapline may not produce any valid data in its Gain Shift block.

Overflow In: Overflow in is an error checking bit from the next higher tapline. If a higher tapline produces an invalid output due to an overflow in the 16-Bit-Adder, the entire chip output will become invalid.

Scan Path Test: The Scan Path Test consists of several inputs and outputs and will be discussed in more detail later on. The related ports are: Scan Path Test Left Out (S_P_Test_Lout), Scan Path Test Right Out (S_P_Test_Rout), Scan Path Test Left In (S_P_Test_Lin), Scan Path Test Right In (S_P_Test_Rin), Shift Right Data In (SRDi), Shift Left Data In (SLDi), Shift Right (SR), and Shift Left (SL).

E. Timing Control

During normal mode of operation, there are a few time-constraints to keep in mind in order to ensure correct results. As shown in Figure 7.20, every new process should start with a synchronous clear. For a synchronous clear all register controls have to be low to force outputs to zero until new data are processed. The next six clock cycles are reserved to load the gain

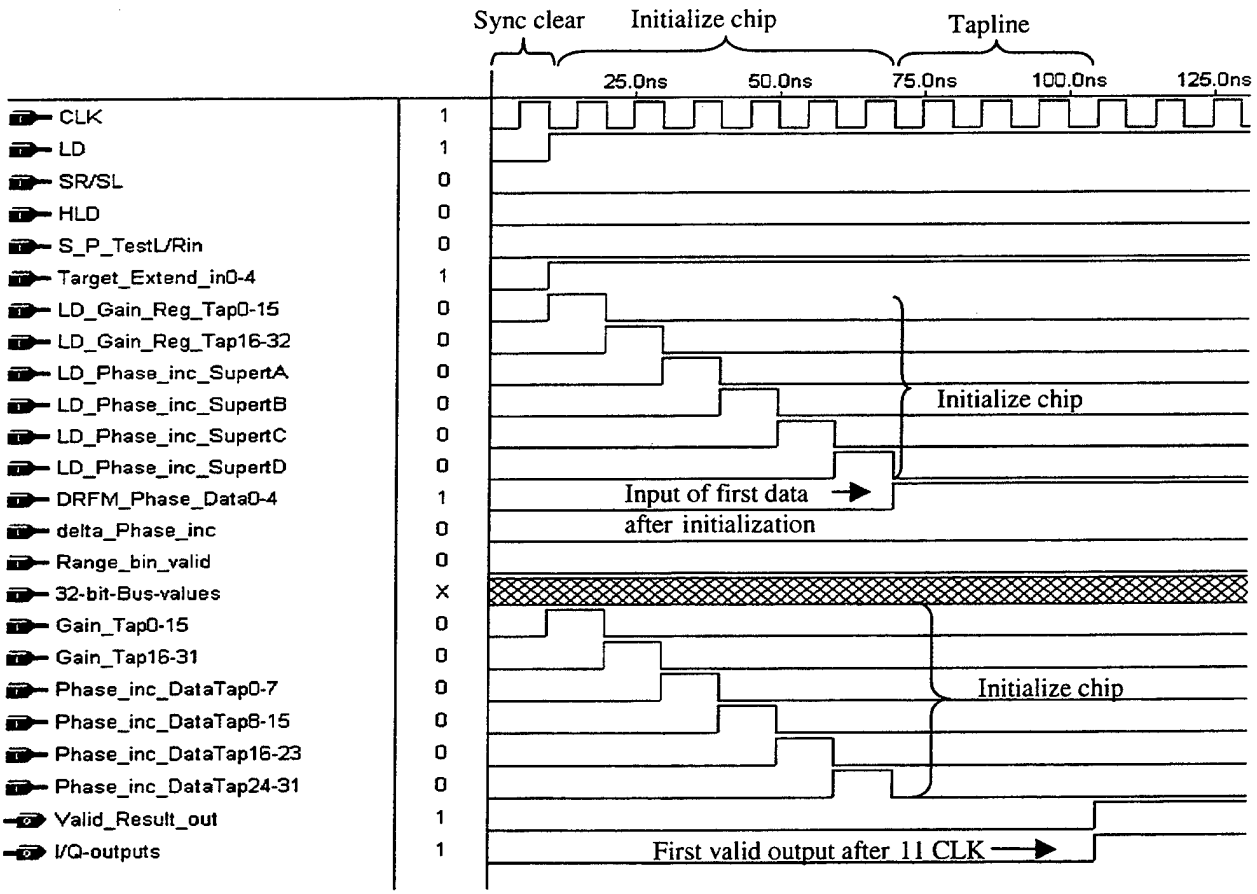


Figure 7.20: Initial loading and first DRFM data inputs

coefficients and the phase increment values through the 32-bit bus. So far seven clock cycles are needed before the first DRFM data can be read into the taplines. Within the taplines, the data treatment takes 4 more clock cycles before the first valid output is present at the first tapline. Thus 11 clock cycles are required before the first valid output. The "valid result out" pad is an indicator for valid results. If this pad goes high, the output is valid, providing "Range bin valid" was set high with the first DRFM sample. After the initialization the DRFM data can be processed within the taplines. Assuming 62 phase samples for a radar pulse and a use of 32 taplines, 31 clock cycles are required to read the 62^{nd} processed sample out of the 32^{nd} tapline. This phase is shown with a low DRFM_Phase_Data0-4, a high Valid_result_out, and high I/Q-outputs. During the time between two radar pulses, new DRFM data cannot be read in. Nevertheless this time interval can be used to rotate the phase increment, as shown in Figure 7.21.

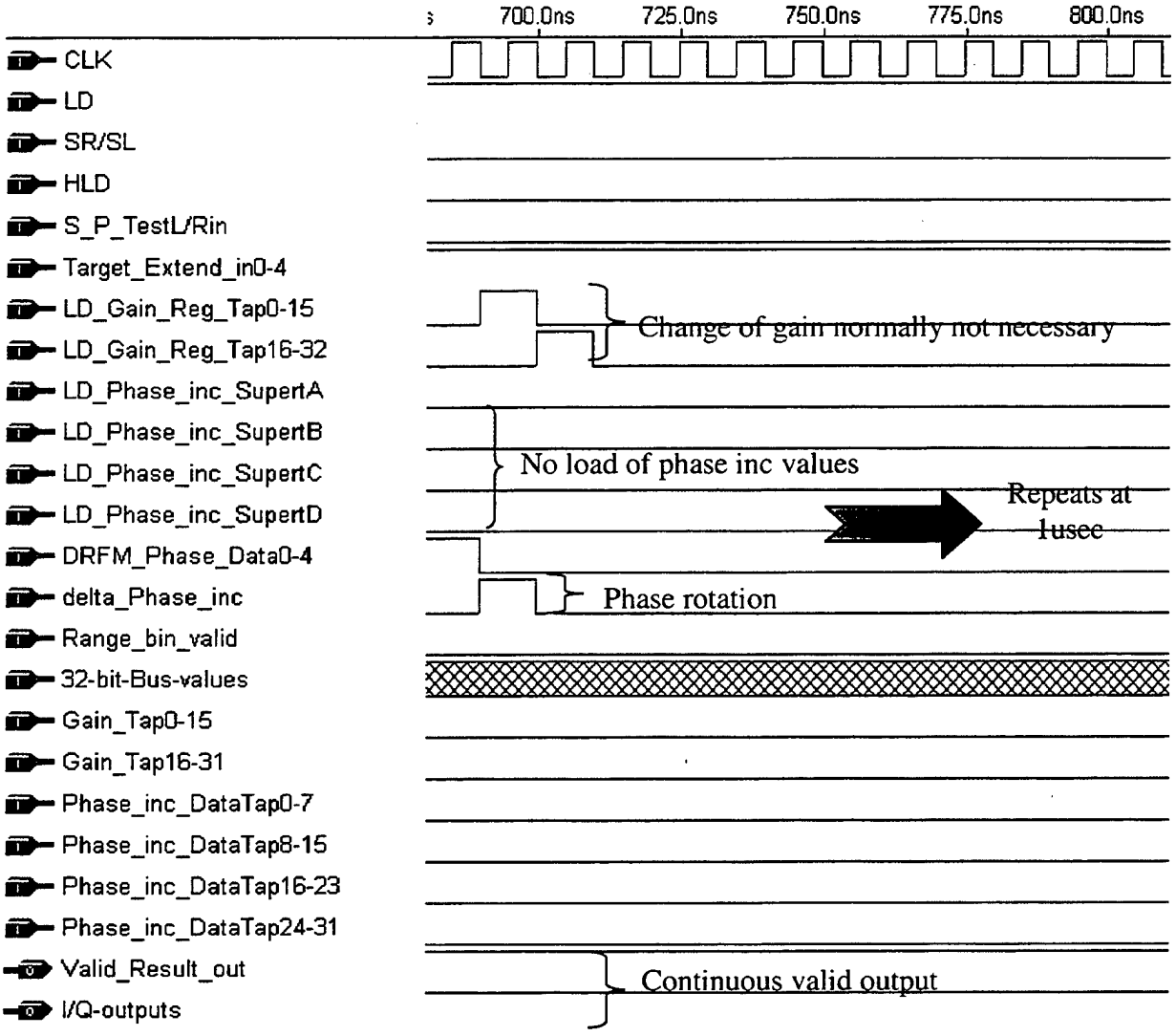


Figure 7.21: Timing diagram between two radar pulses

The process shown between the first two pulses will repeat after every radar pulse. For a regular false target it should not be necessary to change the gain coefficients or the phase increment values. If for some reason a change is required before the last radar pulse of a certain set is processed, the gain coefficients will become effective immediately, whereas a change of the phase increment values will force the phase increment register to clear first and rotate the phase after one full radar pulse is processed. Since this is normally not the case there is no need to be concerned with the gain coefficients and the phase increment values as long as the corresponding control signals are "low".

F. Scan Path Testing

The design of a workable system solution for a given problem is only half of the work. Furthermore one must also be able to test the system to a degree where it can be ensured that the system is fully functional with a high confidence level. In very small-scale digital systems, tests can be performed exhaustively, where the system is exercised over its full range of operating conditions. This method is not an economical or useful approach to verify the functionality. Therefore other strategies are necessary to perform proper testing.

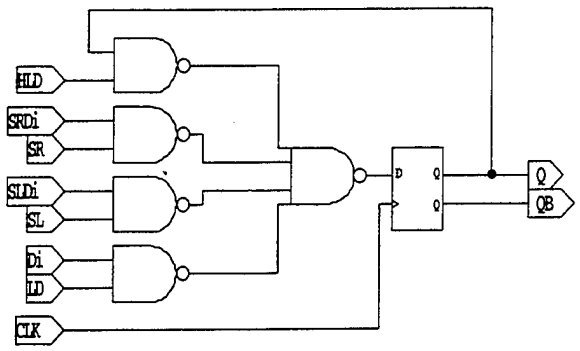


Figure 7.22: Register cell with scan path.

The scan path methodology is probably the most widely used technique for testing those parts of a integrated circuit that are constructed of clocked flip-flops interconnected by combinational logic. As illustrated in Figure 7.22, the scan path can be implemented into a simple circuit very easily. When the circuit is put into test mode, it is possible to shift an arbitrary test pattern into the register. By returning the circuit to normal mode for one clock period, the contents of the scan register and primary input signals act as inputs to the attached combinational circuitry and new values are stored in the by the logic followed scan path register. If the circuit is then placed into test mode again, it is possible to shift out the contents of the scan register for comparison with the correct response.

By using test points, one can easily enhance the absorbability and controllability of a circuit. The scan path register effectively provides such test points, whereas in FPGA design the

implementation of Tristate-buffers was necessary. To control the test points in a scan path test several control signals have to be implemented to adjust the mode of operation. Table 7.6 lists the signals used for the scan path test in the new DIS design.

Padname	Function
SR (Shift Right)	Input pad to control function of register. If high, the data within the register will be shifted to the right with every clock cycle. All other control signal have to be low (HLD, LD, SL)
SL (Shift Left)	Input pad to control function of register. If high, the data within the register will be shifted to the left with every clock cycle. All other control signal have to be low (HLD, LD, SR)
SRDi (Shift Right Data in)	Test data input pad from right front end of scan path test
SLDi (Shft Left Data in)	Test data input pad from left front end of scan path test
SRDo (Shift Right Data out)	Test data output pad for a right shifted output
SLDo (Shift Left Data out)	Test data output pad for a left shifted output

Table 7.6: Scan Path Test Control Signals

A scan path register is a serial cascade of scan path register cells whose inputs and outputs are connected to the internal logic of a chip as illustrated in Figure 7.23. During normal operation, the LD signal is asserted and the logic value at the inputs D0 and D1 reach the outputs Q0 and Q1 after one clock cycle. When the signal SL is asserted during test, the logic value at SLDi arrives at Q1 one clock cycle later and continues propagating to Q0 with the following clock cycle. When the signal SR is asserted during test, the logic value at SRDi arrives at Q0 one clock cycle later and continues propagating to Q1 with the following clock cycle. If the chip is still in test

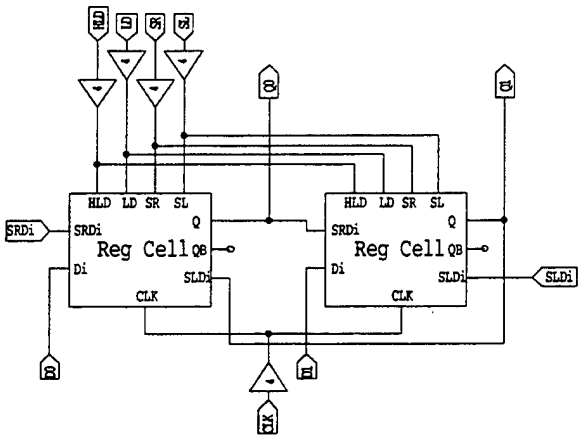


Figure 7.23: 2-bit register.

mode, the values keep propagating in the forced direction through all the connected registers in the scan path.

In summary, the scan path test can be used for two valuable testing functions. First, test data can be placed in every register of the chip to examine the resulting outputs in normal operation mode. Second, after normal operation all stored data in the registers can be read out by shifting to the left or to the right and can be compared with correct results. The scan path test implementation in a tapline is shown in Figure 7.24 below. The path between the taplines within a Supertap and beyond is simply realized by a serial connection of inputs and outputs. The scan chain connects tapline 0 to 31 in a long row of registers. To give an overview about the number of bits penetrating through the scan chain imagine the following calculation:

There are 90 bits used in the registers of a tapline and 32 tapline in the chip. This will result in 90 * 32 = 2.880 values to read out of the entire scan path.

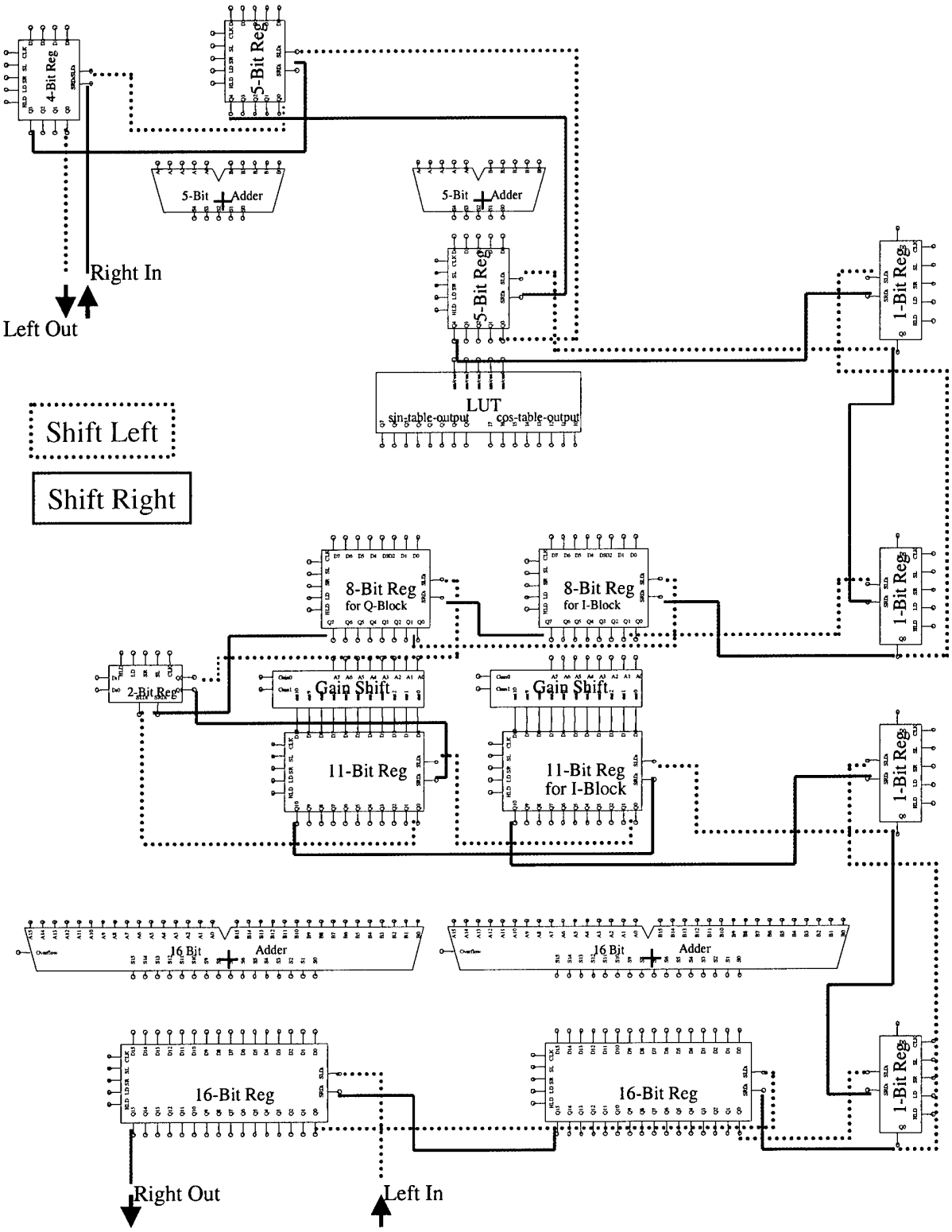


Figure 7.24: Scan path test in a tapline.

G. Simulation in T-Spice

Two goals are established by doing the simulation in T-Spice. First and foremost, the correct logical implementation needs to be verified, which includes the check of each connection between elements (wire connections). The second goal is to prove the proper implementation of the developed algorithm within the circuit. This section describes how the simulation is done in T-Spice. The verification of the circuit functionality is achieved by simulating a smaller part of the entire circuit design and comparing the results to the Matlab simulation.

S-Edit supports a direct export of a schematic layout into a T-Spice readable SPICE format. The exported SPICE file contains only circuit information, but does not contain test-commands or test-vectors. Therefore several lines of code have to be added to create a valid simulation file that can be used in T-Spice. To illustrate the test concept in T-Spice a 2-bit register is used as example. Table 7.7 contains parts of the 2-bit-Register SPICE file that are used for simulation.

T-Spice Code	Meaning
Vdd Vdd Gnd DC 5	Defines the voltages between 0V (Ground) and +5V DC
.include "D:\Chris\Thesis\schematics\testfiles\ Register\2Bit\input_table2Reg.md"	Reads the file input_table2Reg.md, which is an text file containing all input used during simulation
.options prtdel=80n	The option command customizes the simulation. PRTDEL sets the reading for output pads to exact every 80nsec
.tran 10n 800n start=70n	Performs a transient analysis with a maximum step size for calculations of 10nsec, a simulation stop time of 800nsec and an offset for the first output reading of 70nesc
.print tran "D:\Chris\Thesis\Schematics\testfiles\ Register\2Bit\Inputs.out" V(CLK) V(SL) V(SR) V(SLDi) V(SRDi) V(LD) V(HLD) V(D0) V(D1) .print tran "D:\Chris\Thesis\Schematics\testfiles\ Register\2Bit\Outputs.out" V(Q0) V(Q1)	The print tran command is used to define the monitored output pads and the file in which the records are saved. The file "inputs.out" records all control signals and the inputs of the register, whereas the file "Outputs.out records only the outputs of the register.
.param l=0.05u	Sets the wavelength to 0.05μm
.include "D:\Chris\Thesis\ModelParammod.md"	Includes the transistor parameters for the target process (MOSIS – HP 0.5μm) used for the simulation.

Table 7.7: T-Spice Simulation Commands

T-Spice is not a logical circuit simulator, but can perform various analog simulations like DC-analysis and frequency sweeps. Nevertheless T-Spice can make use of the "bit"-command that offers to push binary inputs into the input pads of the circuit representation. The voltages are 0V for a logical zero and 5V for a logical one. By defining the inputs as voltage sources, T-Spice analyses the input vectors, calculates a DC operating point, and calculates the

defined output pads in form of voltages. Below is an example of the input vectors for the 2-bit-Register simulation.

```
VinD0 D0 Gnd bit       ({0010111111} on=5.0 off=0.0 pw=80n rt=0.1n ft=0.1n
VinD1 D1 Gnd bit       ({0001011111} on=5.0 off=0.0 pw=80n rt=0.1n ft=0.1n
VinSRDi SRDi Gnd bit   ({0000000000} on=5.0 off=0.0 pw=80n rt=0.1n ft=0.1n
VinSLDi SLDi Gnd bit   ({0000000011} on=5.0 off=0.0 pw=80n rt=0.1n ft=0.1n
VinHLD HLD Gnd bit     ({0000100000} on=5.0 off=0.0 pw=80n rt=0.1n ft=0.1n
VinLD LD Gnd bit       ({0111010000} on=5.0 off=0.0 pw=80n rt=0.1n ft=0.1n
VinSR SR Gnd bit       ({0000001100} on=5.0 off=0.0 pw=80n rt=0.1n ft=0.1n
VinSL SL Gnd bit       ({0000000011} on=5.0 off=0.0 pw=80n rt=0.1n ft=0.1n
VinCLK CLK Gnd bit         ({01}  on=5.0 off=0.0 pw=40n rt=0.1n ft=0.1n
```

The input vectors are defined by a name for the voltage source (input pad) against ground, a bit pattern used as inputs for the voltages sources, the definition of zero and one, a pulse width (pw) of the signal, a rise time (rt), and a fall time (ft) in nano seconds. The registers are constructed for the use of a positive edge triggered clock. This means that all signals have to be present before the clock changes from low to high. A change of a value after the clock goes high cannot be processed properly. As an illustrated example in Figure 7.25, the clock starts with a "low" of 40ns and changes to "high" afterwards. Therefore, the entire clock cycle is 80ns, which corresponds to the pulse width of the input signals.

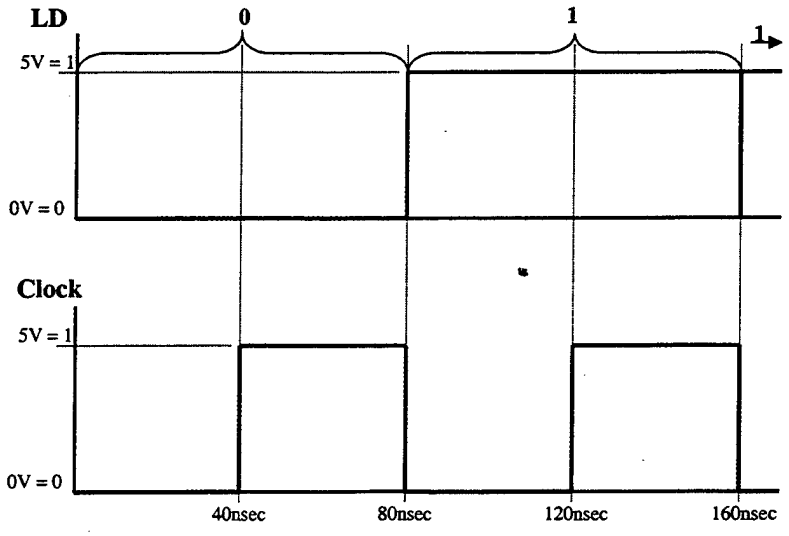


Figure 7.25: Signal-clock concept

The above-mentioned input values are used to test the behavior of the 2-bit-Register under normal and test mode conditions. Table 7.8 illustrates the basic test concept and the relation between the control signals.

D0	SRDi	SLDi	HLD	LD	SR	SL	CLK	Q0	Remark
0/1			0	0	0	0	0→1	0	Synchronous Clear
0/1			0	1	0	0	0→1	0/1	Data (D0) – normal operation
0/1			1	0	0	0	0→1	0/1	Previous data, "do nothing"
	0/1		0	0	1	0	0→1	0/1	Right data (SRDi)
		0/1	0	0	0	1	0→1	0/1	Left Data (SLDi)

Table 7.8: Test Concept of the 2-Bit-Register

The test mode signals are in direct relationship to each other, because only one input of SR, SL, HLD, and LD can be high at a time to perform a legal operation in test mode.

During the transient simulation, T-Spice uses the input vectors to determine the voltage values for the outputs. The determined values are saved in the predefined files and the transient analysis results are automatically stored in a separate file. These transient results can be used to go back to the circuit in S-Edit and "probe" in the schematic layout. The probing will call W-Edit automatically and create a graphical output of the voltage in time for the probed node. The user defined output files, which contains the simulation results holds exact voltage values in the region between 0V and 5V, as shown below in Table 7.9.

Time (sec)	v(Q0) (Volts)	V(Q1) (Volts)
7.0000E-08	1.2408E-07	1.2414E-07
1.5000E-07	1.4298E-07	1.4302E-07
2.3000E-07	5.0000E+00	6.3223E-08
3.1000E-07	8.6684E-08	5.0000E+00
3.9000E-07	1.9326E-07	5.0000E+00
4.7000E-07	5.0000E+00	5.0000E+00
5.5000E-07	2.8934E-07	5.0000E+00
6.3000E-07	1.2812E-07	-2.4128E-07
7.1000E-07	4.3775E-08	5.0000E+00
7.9000E-07	5.0000E+00	5.0000E+00
8.0000E-07	5.0000E+00	5.0000E+00

Table 7.9: Output Table for the Transient Analysis of a 2-bit-Register

The analog output values distinguish between one and zero. The output values are analog voltages and the results have to be sent through a hard limiter to get a binary output table in order to compare the results with the correct binary output pattern produced by Matlab.

In very small-scale digital systems, tests can be performed exhaustively, where the system is exercised over its full range of operating conditions. This kind of test was used to simulate the elements of level 1 and 2 of the design hierarchy, as exercised for the 2-bit-Register test case. However, for higher-level elements, the input values were chosen more carefully, to simulate only critical cases. By increasing the number of sub circuits, the simulation time and the amount of required computing power increases in an almost exponential manner. For a Supertap simulation (level 4), a PC with Pentium III processor and 768MB RAM could not satisfy the need for resources by the T-Spice simulation.

Two Tap Simulation

Due to the limitations in available computer resources, it was decided to prove the algorithm with a two tapline circuit, as shown in Figure 7.26. The simulation takes approximately

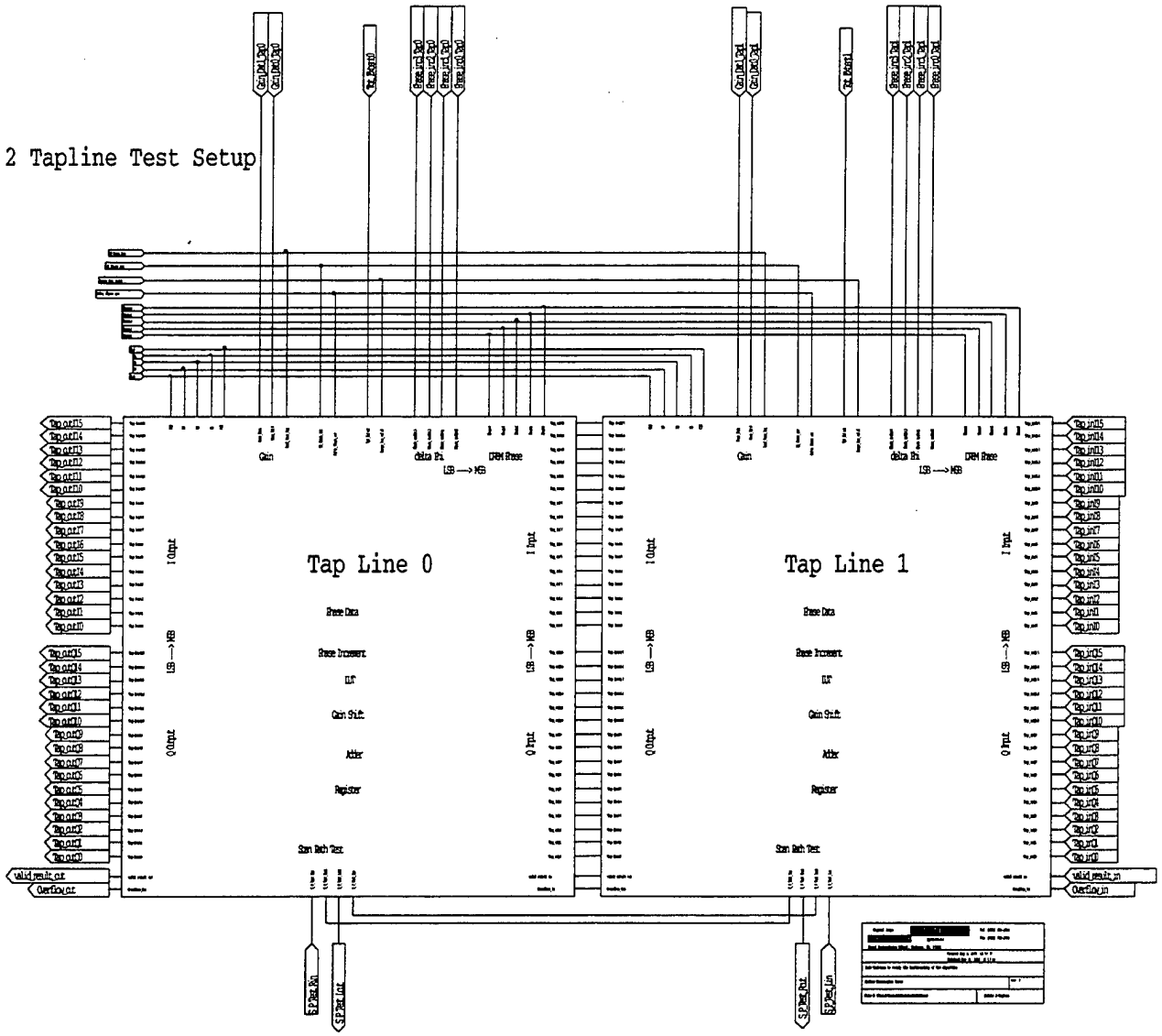


Figure 7.26: 2-Tapline-test circuit

18 hours and provides the results shown below. The Matlab programs discussed earlier were used to create DRFM phase data for 10 radar pulses with 15 samples each. These data were inputs for the simulation in T-Spice for the 2-tapline-circuit representation. The simulation results were used as inputs to the "hard_limiter2Taps.m" Matlab program to prepare the data for further treatment and bring them into a binary form. The resulting data were translated into integers and plotted in Matlab. The correct values were obtained from a Matlab simulation. The following

figures and tables give an overview about the data processing. The Matlab created DRFM phase data are listed in the simulation input file and were used as input vectors for the simulation in T-Spice. Matlab used the same input vectors. The simulation vector table contains approximately 160 input vectors (150 DRFM phase data plus control signals) for every input pad shown in Figure 7.26 and is only listed in the Appendix.

The range-Doppler-amplitude map entry program was modified for the 2-tapline test case in so that only 10 radar pulses with 15 samples per pulse could be used in order to decrease the simulation time in the hardware. For this setup, 150 DRFM phase data samples have

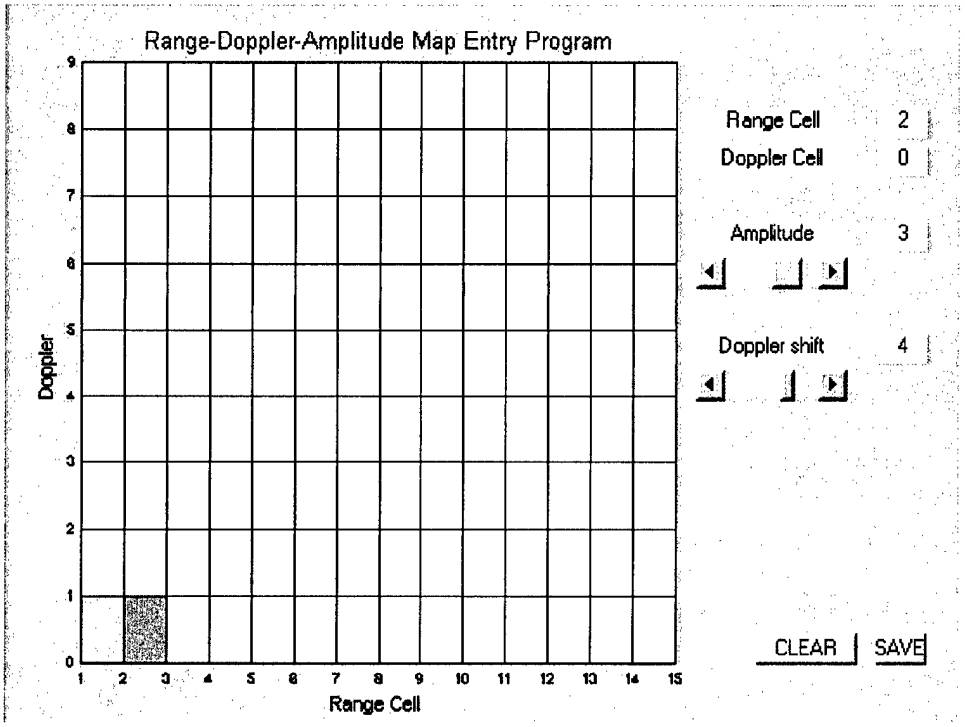


Figure 7.27: Range-Doppler-Amplitude Map Entry Program

been produced in Matlab. Since 2 taplines were used, only two cells in the range-Doppler map are defined for the false target generation. The setup for the amplitude (gain coefficients in hardware) and the Doppler Shift (phase increment values in hardware) are summarized in Table 7.10.

Target Cell	*Range Cell*	*Doppler Cell*	*Amplitude*	*Doppler Shift*	*Remark*
1	1	1	2	2	Tap 0 – 1st Tap
2	2	1	3	4	Tap 1 – 2nd Tap

Table 7.10: Matlab Inputs in the Range Doppler Map

The rest of the simulation in Matlab follows the same path as described in Chapter 4. The values in Table 7.10 can be translated into T-Spice test vectors and specify the input parameters to the taplines, as listed in Table 7.11.

tapline	*Range Cell*	*Doppler Cell*	*Gain coefficients*	*Phase Increment*	*Remark*
0	-	-	01	0010	Binary inputs
1	-	-	10	0100	Binary inputs

Table 7.11: T-Spice Inputs in the Range-Doppler Map

Table 7.12 shows only a small part of the input vectors used for this test case, but they serve to explain the interaction between input control signals, input vectors and the output values.

Input Pad Name	**Sync Clear**	**Radar Pulse 1**		**Radar Pulse 2**		**Radar Pulse 3**
Phase0	0	001101010000110	0	001101010000110	0	001101010000110
Phase1	0	000100000011111	0	000100000011111	0	000100000011111
Phase2	0	001110100100110	0	001110100100110	0	001110100100110
Phase3	0	000101010011111	0	000101010011111	0	000101010011111
Phase4	0	000001110100010	0	000001110100010	0	000001110100010
Delta Phase inc	0	000000000000000	1	000000000000000	1	000000000000000
LD Phase inc	0	100000000000000	0	000000000000000	0	000000000000000
LD Gain	0	100000000000000	0	100000000000000	0	100000000000000
Range bin valid		111111111111111	0	111111111111111	0	111111111111111

Table 7.12: Extract of Input Vectors for the 2-tapline-Test

Note that the Phase 0-4 represents the 5-bit DRFM phase data input to both taplines. Also, the first input performs a synchronous clear to zero all registers. A radar pulse consists of 15 samples, where each sample is read in with every clock cycle. Note also that the gap between the radar pulses is manually set and not part of the DRFM data.

Three radar pulses are shown with 15 DRFM phase data samples. Between the radar pulses are gaps to load zeros for the "range bin valid" control signal and to increment the "rotating phase" by one rotation The gap is necessary to achieve the delay in the 16-bit adder chain between the taplines. Recall, that when the last sample is processed through the taplines the output is the sum of the last output in tapline 0 added with the last output of tapline 1. Therefore

one more clock cycle is required to read out the last final output, which is the last processed pulse in tapline 1 added to a zero from tapline 0. This produces the 16th output value for only 15 input samples. The binary word of zero in tapline 0 is achieved by setting the "range bin valid" control bit to low for one clock cycle to clear the Pipeline Reg 3 in tapline 0 and tapline 1. After the last processed phase sample in tapline 1 reaches the final output through tapline 0, the first fully processed sample from the next radar pulse is already present for output. Due to the low "range bin valid" for one clock cycle, the Pipeline Reg 3 were cleared (in both taplines), so that the output of tapline 1 was zero. Therefore the final output is just the sum of the output of tapline 0 and a zero valued binary word from tapline 1.

CLK	Valid Result	I values	Q values	Pulse#	Sample#
1	0	0000000000000000	0000000000000000	Sync	Clear
2	0	0000000000000000	0000000000000000		
3	0	0000000000000000	0000000000000000		
4	0	0000000000000000	0000000000000000		
5	1	0000000011111110	0000000000000000	1	1
6	1	0000001011111010	0000000000000000	1	2
7	1	0000001010000010	0000000011010110	1	3
8	1	0000000000010000	0000000111000100	1	4
9	1	1111111010110110	0000000011101000	1	5
10	1	0000000110110100	0000000010000010	1	6
11	1	0000000000010110	1111110101011100	1	7
12	1	1111111100100100	1111110110000010	1	8
13	1	0000000110101110	1111111000100100	1	9
14	1	0000000101100010	1111111100111000	1	10
15	1	1111111001011110	1111111101010100	1	11
16	1	1111111010110110	0000001010101100	1	12
17	1	1111111000101000	0000000111100000	1	13
18	1	1111111100000110	0000000000110000	1	14
19	1	0000000110001110	0000000011100100	1	15
20	1	1111111100100100	0000000111001000	1	15
21	1	0000000011101000	0000000001100100	2	1
22	1	0000001001000100	0000000111010100	2	2

Table 7.13: T-Spice Simulation Outputs (hard limited)

Table 7.13 shows only the first 22 clock cycles out of 171. The first four clocks are used to process the first sample within the taplines. Bus cycles are not needed for a 2-tapline-test, but have to be included for a 32-tapline-test. The I and Q values are in form of: most significant bit → least significant bit. Table 7.13 lists the output I and Q output values corresponding to input

DRFM phase data. The first 5 clock cycles are needed to process the first input, where the first output is a synchronous clear. Clock 19 and 20 are the last outputs from radar pulse 1. Due to the delay just described, sample 15 will produce two outputs. The phase outputs are 16-bit 2-complemet words that are translated into a decimal representation to be processed in Matlab.

To extend the 2-tapline-test case to x-number of taplines, it is requires to set the "range bin valid" control bit to at least x-1 low inputs (0's) between two radar pulses and after the last radar pulse in order to get the correct delay in the adder chain. Furthermore the delay produces outputs in the range of

$$\text{number of DRFM samples per pulse} + (x-1) = \text{number of outputs.} \qquad (7.5)$$

After transforming the T-Spice simulation results into a decimal representation, they can be processed in Matlab. Figure 7.28 shows two-dimensional contour plots for the Matlab and the T-Spice simulation results in comparison. By visual inspection there is no obvious difference in the preliminary simulation results.

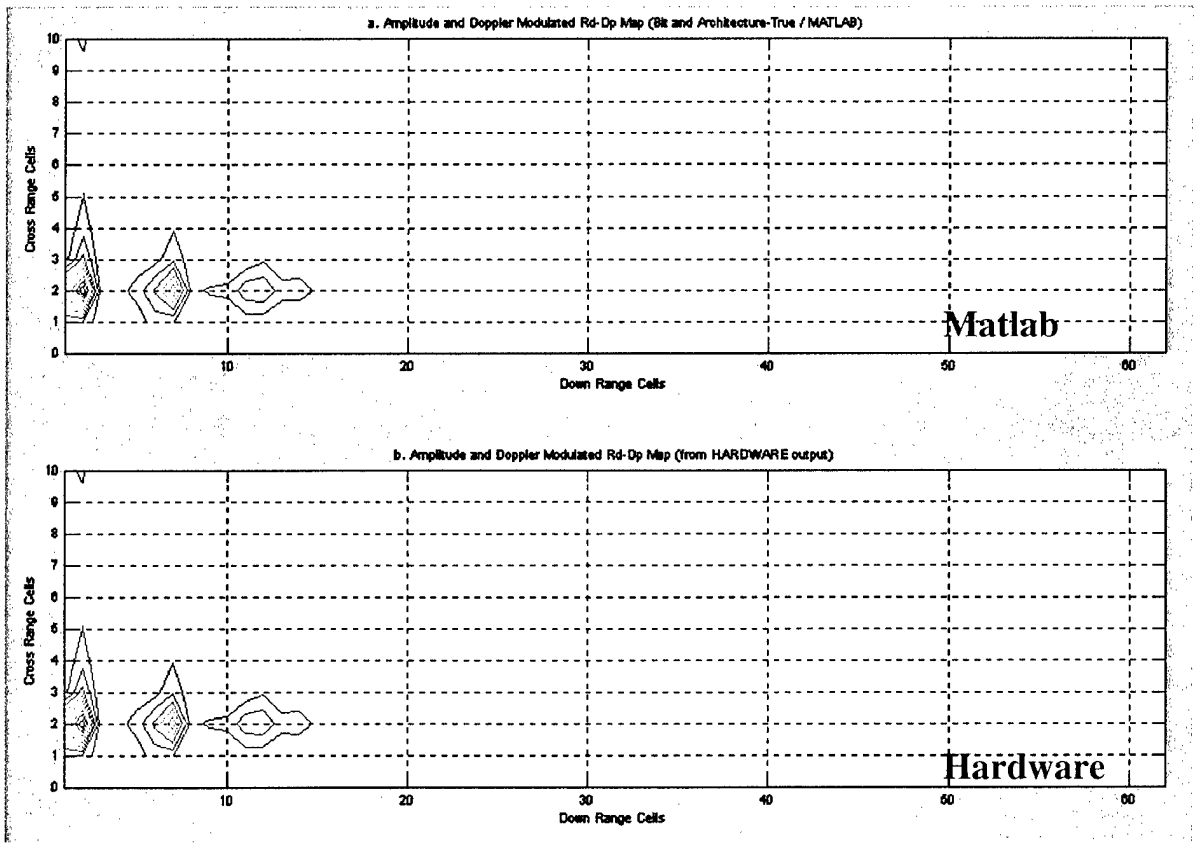


Figure 7.28: 2D plot of simulation results

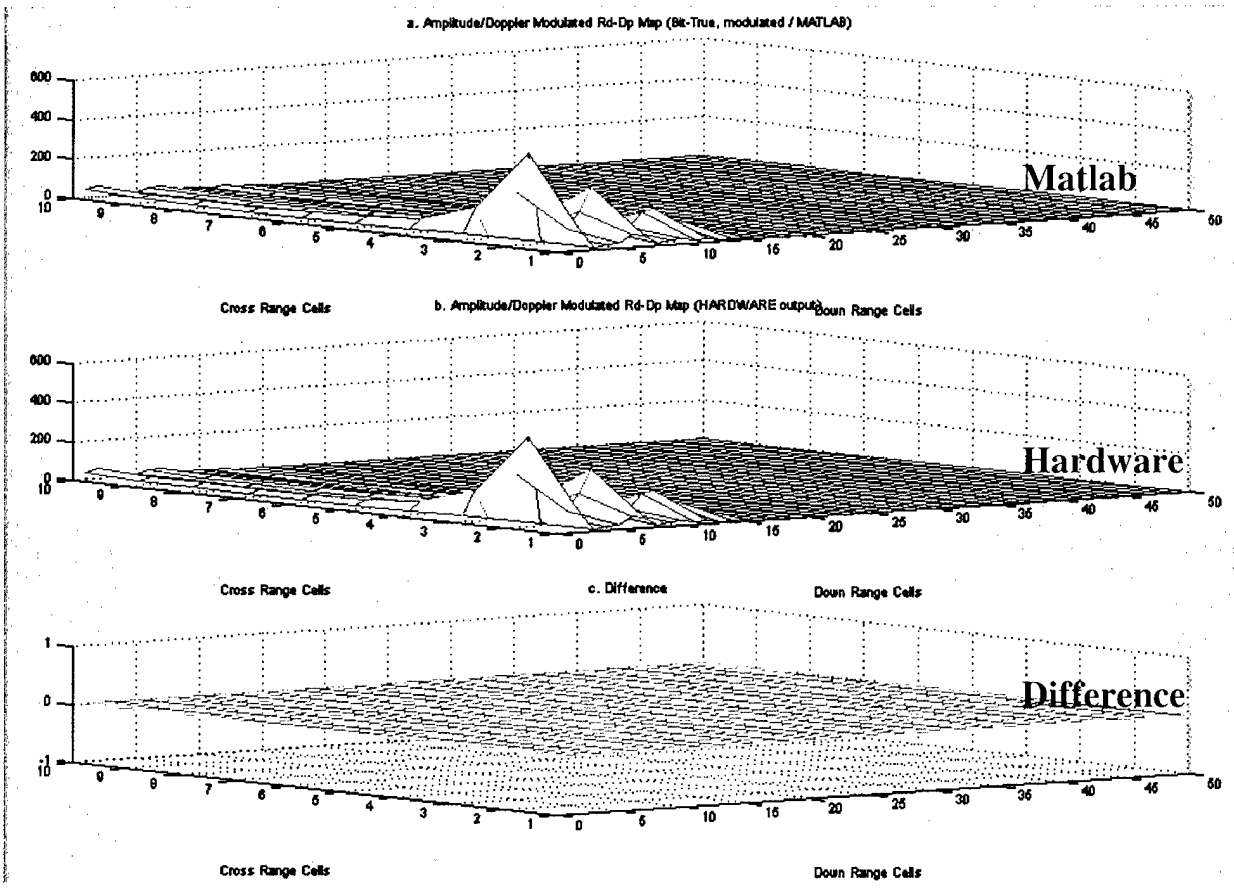


Figure 5.29: 3D plot of simulation results and comparison

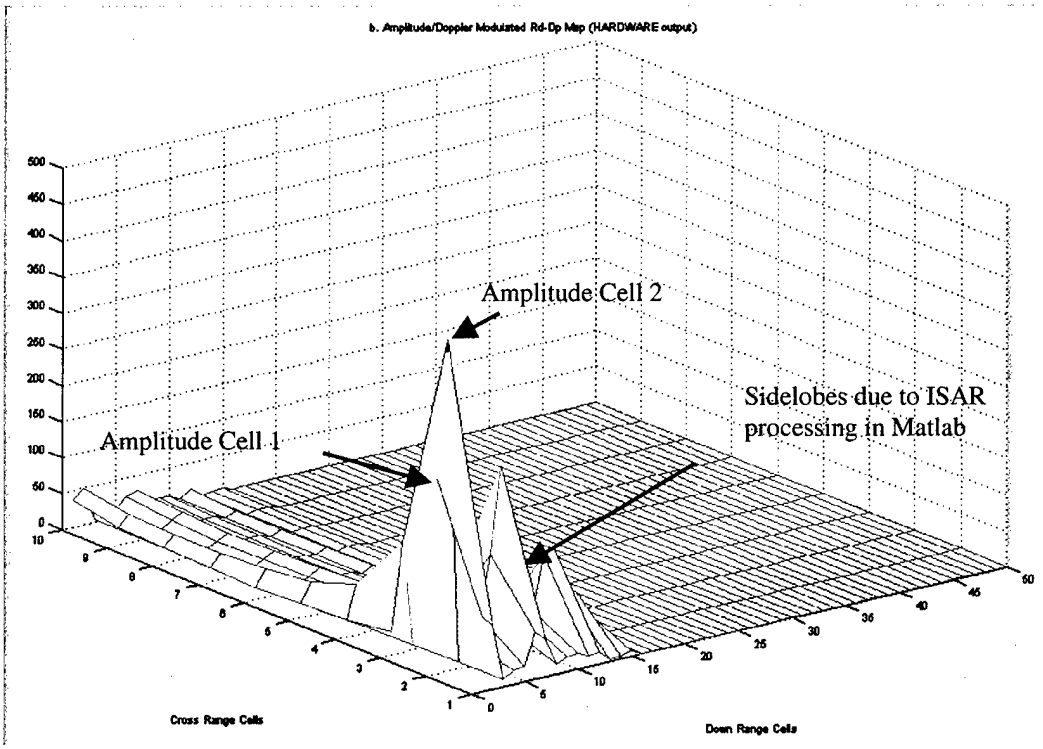


Figure 7.30: Hardware results

Figure 7.29 shows the three-dimensional representation of the results and a comparison between the T-Spice and Matlab simulation. The difference is calculated by subtracting the Matlab simulation output data from the T-Spice simulation output data. Since the difference is exactly zero, the proof is complete. The Matlab simulation and the T-Spice simulation process the data in the same way and therefore represent the same algorithm.

To extend the testing for a Supertap and the top-level design, other tools are required. GateSim is part of the Tanner Tool Environment and is a logic simulator in MS-DOS. S-Edit can create a file type readable by Nettran, which can be imported into GateSim. Until now, this has not been investigated but is on focus for future studies.

References

[1] Donald R. Wehner, "High Resolution Radar", 2nd Edition.

[2] R. M. Nuthalapathi, "High Resolution Reconstruction of ISAR Images", IEEE Transactions on Aerospace and Electronic Systems", Vol. 28, No. 2, p.462ff, April, 1992.

[3] P. E. Pace, Surratt, R. E., Yeo, S.-Y., "Signal Synthesizer and Method Therefor," Patent File Attorney Docket No. 79,429, Sept. 1, 1999.

[4] T. T. Vu, et al., "A GaAs Phase Digitizing and Summing System for Microwave Signal Storage," *IEEE Journal of Solid State Circuits*, Vol. 24, p. 104, February, 1989.

[5] Mathwork Inc., Homepage for MATLAb, http://www.mathworks.com.

[6] Yeo, Siew-Yam, "A Digital Image Synthesizer for ISAR Counter-Targeting," Master's Thesis, Naval Postgraduate School, Monterey, September 1998.

[7] MAX+PLUS II Getting Started version 8.1 (5.4 MB).

[8] Altera Max+Plus II Online-manual.

[9] Altera Homepage, (http://www.altera.com/).

[10] Visual Software Solutions Inc. Homepage, http://www.statecad.com.

[11] Visual Software Inc., Statecad 5.0 and Statebench printed manuals.

[12] SimGen Online manual.

[13] Mentor Graphics Homepage, http://www.mentor.com.

[14] AMI FPGA/ASIC Design Techniques Seminar, April 16, 1999.

[15] Tanner Tools, printed manuals for LVS 8.02, Nettran, general instructions for Tanner Tools.

Table of Abbreviation

ADC	Analog to Digital Conversion
AHDL	Altera Hardware Description Language
AMI	American Microsystems Inc.
ASCM	Anti Ship Cruise Missile
ASIC	Application-Specific Integrated Circuit
BIT	Built In Test
CMOS	Complementary Metal Oxide Semiconductor
CPI	Coherent Processing Interval/Synthetic Aperture Frametime
CPLD	Complex Programmable Logic Device
DBS	Doppler Beam Sharpening
DIS	Digital Image Synthesizer
DRFM	Digital Radio Frequency Memory
DSP	Digital Signal Processing
EA	Electronic Attack
EAB	Embedded Array Block
EW	Electronic Warfare
FFT	Fast Fourier Transform
FLEX	Flexible Logic Element Matrix
FPGA	Field Programmable Gate Array
GUI	Graphical User Interface
HDL	Hardware Description Language
HRR	High Resolution Radar
IFFT	Inverse Fast Fourier Transform
ISAR	Inverse Synthetic Aperture Radar
LOS	Line Of Sight
LPM	Library of Parameterized Module
LUT	Look-Up Table
NRL	Naval Research Laboratory
PC	Personal Computer
PLD	Programmable Logic Device

PRI	Pulse Repetition Interval
RCS	Radar Cross Section
RF	Radio Frequency
RRC	Radar Resolution Cell
SAR	Synthetic Aperture Radar
TDL	Tapped Delay Line
VHDL	Very High Speed Integrated Circuit Hardware Description Language

Appendix A - MATLAB Codes

Version 1

runDISv1.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% runDISv1.m
% This script file will help you to run the Digital Image Syntesizer (DIS)
% This is a modified version that is able to handle different target extents
% (that is, how many taps the user would like to use that will represent the
% radial length of the target, seen from the ISAR)
% Created by:
%   MAJ Stig Ekestorm, Sep -99
%   Naval Postgraduate School
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%set path
%cd c:\temasek\denise\thesis\final_design\vbfiles

%clear the workspace
clear

%run the graphical user interface to specify target parameters
guiv1
disp('Enter the values in the Grapical User Interface')
disp('Press any key to continue')
pause

%pre-process signal parameters, simulate ISAR
mathostv1
disp('Press any key to continue')
pause

%simulate the DIS in Matlab
simhwchkv1
disp('Press any key to continue')
pause

%see to that data from hardware are available
disp('This should be the latest time to see to that data from hardware')
disp('are available in the two files imagei.txt and imageq.txt')
disp(' ')
disp('When this is taken care of, return to the MATLAB Command Window')
disp('and press any key to continue')
pause

%plot results for visual comparison
plothwv1

%end of file
```

guiv1.m

```
function [dat] = guiv1(action);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% guiv1.m
% Get inputs from screen
% SY YEO, 30 Jan 98
% Modified by Stig Ekestorm, Aug -99
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

global hf
global h1
global h2
global data
global loc
global patchsize
global txtloc
global count
global ph

if nargin<1,
    action='start';
 end;

if strcmp(action,'start'),

   % initialize the figure

   set(0,'DefaultAxesFontSize',6);

   hf = figure(1); clf
   set(hf,'NextPlot','add');

   set(hf, ...
      'NumberTitle','off', ...
      'Name','Naval PostGraduate School',...
      'backingstore','off',...
      'Units','normalized');

   rg_pts = 62;
   dp_pts = 64;
   data = [];  loc = [];
   count = 0;
   ph = [];

   h1 = axes('Position',[0 0 1 1],'Visible','off');
   h2 = axes('Position',[0.1 0.1 0.6 0.8]);

   set(hf,'currentaxes',h2);

   xa = 1:rg_pts;
   ya = 0:(dp_pts-1);

   xtick = 0:1:rg_pts;
   set(gca,'XTickMode','manual');
```

```
set(gca,'XLimMode','manual');
set(gca,'XLim',[1 rg_pts]);
set(gca,'XTick',xtick);
set(gca,'XGrid','on');
set(gca,'GridLineStyle','-');

set(gca,'YTickMode','manual');
set(gca,'YLimMode','manual');
set(gca,'YLim',[0 dp_pts-1]);
ytick = 0:1:dp_pts;
set(gca,'YTick',ytick);
set(gca,'YGrid','on');
set(gca,'GridLineStyle','-');

xh = xlabel('Range Cell'); set(xh,'FontSize',8); clear xh
yh = ylabel('Doppler'); set(yh,'FontSize',8); clear yh
ht = title('Range-Doppler-Amplitude Map Entry Program');
set(ht,'FontSize',10,'Color',[0 0 1]);

a = uicontrol('Units','normalized', ...
   'BackgroundColor',[.9 .9 .9], ...
   'Position',[0.72 0.80 0.15 0.04], ...
   'Style','text', ...
   'FontSize',6,...
   'String','Range Cell', ...
   'Tag','aText');

b = uicontrol('Units','normalized', ...
   'BackgroundColor',[.9 .9 .9], ...
   'Position',[0.72 0.75 0.15 0.04], ...
   'Style','text', ...
   'FontSize',6,...
   'String','Doppler Cell', ...
   'Tag','bText');

c = uicontrol('Units','normalized', ...
   'BackgroundColor',[.9 .9 .9], ...
   'Position',[0.72 0.65 0.15 0.04], ...
   'Style','text', ...
   'FontSize',6,...
   'String','Amplitude', ...
   'Tag','cText');

c11 = uicontrol('Units','normalized', ...
   'BackgroundColor',[.9 .9 .9], ...
   'Position',[0.72 0.60 0.15 0.04], ...
   'Style','slider','min',0,'max',4,...
   'SliderStep',[0.25 0.5],...
   'Callback','guiv1(''update1'')');

d = uicontrol('Units','normalized', ...
   'BackgroundColor',[.9 .9 .9], ...
   'Position',[0.72 0.50 .15 0.04], ...
   'Style','text', ...
   'FontSize',6,...
   'String','Doppler shift');
```

```
d11 = uicontrol('Units','normalized', ...
   'BackgroundColor',[.9 .9 .9], ...
   'Position',[0.72 0.45 0.15 0.04], ...
   'Style','slider','Min',-10,'Max',10,...
   'SliderStep',[0.05 0.1],...
   'Callback','guiv1(''update1'')');

a1 = uicontrol('Units','normalized', ...
   'BackgroundColor',[1 1 1], ...
   'Position',[0.9 0.80 0.05 0.04], ...
   'Style','text', ...
   'FontSize',6,...
   'String','', ...
   'Tag','a1Text');

set(gcf,'currentaxes',h1);
b1 = uicontrol('Units','normalized', ...
   'BackgroundColor',[1 1 1], ...
   'Position',[0.9 0.75 0.05 0.04], ...
   'Style','text', ...
   'FontSize',6,...
   'String','', ...
   'Tag','a2Text2');

set(gcf,'currentaxes',h1);
c1 = uicontrol('Units','normalized', ...
   'BackgroundColor',[1 1 1], ...
   'Position',[0.9 0.65 0.05 0.04], ...
   'Style','text', ...
   'FontSize',6,...
   'Callback','guiv1(''update'')',...
   'String','');

d1 = uicontrol('Units','normalized', ...
   'BackgroundColor',[1 1 1], ...
   'Position',[0.9 0.50 0.05 0.04], ...
   'Style','text', ...
   'FontSize',6,...
   'Callback','guiv1(''update'')',...
   'String','');

q1 = uicontrol('Units','normalized', ...
   'BackgroundColor','Yellow', ...
   'Position',[0.9 0.10 0.05 0.04], ...
   'Style','pushbutton', ...
   'FontSize',8,...
   'String','SAVE', ...
   'Callback','guiv1(''savequit'')');

q2 = uicontrol('Units','normalized', ...
   'BackgroundColor','Yellow', ...
   'Position',[0.78 0.1 0.1 0.04], ...
   'Style','pushbutton', ...
   'FontSize',8,...
   'String','CLEAR', ...
   'Callback','guiv1(''start'')');
```

```
    txtloc = [a a1 b b1 c c1 c11 d d1 d11];
    % Assign action when mouse button is pressed
    set(h2,'ButtonDownFcn','guiv1(''down'')');

elseif strcmp(action,'down'),
    % Obtain coordinates of mouse click location in axes units

    set(hf,'currentaxes',h2);
    pt=get(h2,'currentpoint');

    x=pt(1,1); xf = floor(x);
    y=pt(1,2); yf = floor(y);
    [r,c] = size(data);

    set(txtloc(7),'Value',0);
    set(txtloc(9),'Value',0);

    tmp = [x y 1 0];
    loc = [loc tmp];
    tmp = [xf yf 1 0];
    data = [data;tmp];
    [r,c] = size(data);
    ypos = [yf yf+1 yf+1 yf];
    xpos = [xf xf xf+1 xf+1];

    count = count + 1;
    %disp(count);
    txt = ['Tag',num2str(count)];
    ptr = patch(xpos,ypos,[1 1 1]*0.9);
    %disp(ptr);
    set(ptr,'ButtonDownFcn',[ ...
          'guiv1(''update'')']);
    set(ptr,'Tag',txt);
    set(ptr,'UserData',[xf yf 1 0]);
    ph = ptr;
    set(txtloc(2),'String',xf);
    set(txtloc(4),'String',yf);
    set(txtloc(6),'String',1);
    set(txtloc(9),'String',0);

elseif strcmp(action,'update'),
    % Determine the patch that is selected
    ph = gcbo;
    %set(ph,'Selected','on');
    % Retrieve the values for that patch and display it
    % txtloc = [a a1 b b1 c c1 c11 d d1 d11];
    % txtloc 2: Range cell
    % txtloc 4: Doppler cell
    % txtloc 6: Amplitude  txtloc 7: Slider bar
    % txtloc 9: Doppler offset  txtloc 9: Slider bar
    ud = get(ph,'UserData');
    set(txtloc(2),'String',ud(1));
    set(txtloc(4),'String',ud(2));
    set(txtloc(6),'String',ud(3));
    set(txtloc(9),'String',ud(4));
    set(txtloc(7),'Value',ud(3));
    set(txtloc(10),'Value',ud(4));
```

```
  elseif strcmp(action,'update1'),
     if (~isempty(ph))
        ph1 = gcbo;
        if ((ph1 == txtloc(7)) | (ph1 == txtloc(10)))
           ud = get(ph,'UserData');
           xf = ud(1);   yf = ud(2);
           ypos = [yf yf+1 yf+1 yf];
           xpos = [xf xf xf+1 xf+1];
           set(ph,'Selected','off');
           % Update the amplitude/doppler values
           if (ph1 == txtloc(7))
              tmp1 = get(txtloc(7),'Value');
              tmp1 = round(tmp1)
              set(txtloc(6),'String',tmp1);
              set(txtloc(7),'Value',tmp1);
              if (tmp1 < 1),
                 set(txtloc(7),'Value',1);
                 set(txtloc(6),'String','1');
                 tmp1 = 1;
              end
              col = [1 1 1]*(1-tmp1/10);
              set(ph,'FaceColor',col);
              set(ph,'UserData',[ud(1) ud(2) tmp1 ud(4)]);
           end
           if (ph1 == txtloc(10))
              tmp2 = round(get(txtloc(10),'Value'));
              set(txtloc(9),'String',tmp2);
              set(txtloc(10),'Value',tmp2);
              set(ph,'UserData',[ud(1) ud(2) ud(3) tmp2]);
           end
           %disp('HHH');
           %disp(get(ph,'Tag'))
           %disp(get(ph,'UserData'))
        end
     end

elseif strcmp(action,'savequit'),
   dat = [];
   for i = 1:count
      tt = findobj('Tag',['Tag' num2str(i)]);
      tmp = get(tt,'UserData')
      dat = [dat;tmp];
      fprintf('count = %d, Tag = %s  ',count,get(tt,'Tag'));
      disp(tmp);
   end
   save -ascii sigpar1 dat
   close gcbf
end
```

mathostv1.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%mathostv1.m
%
% Modified version of mathostv0.m
% Modified by Stig Ekestorm, Aug -99
%
% mathostv0.m  Generate pri_dp map and range-doppler map
% - generates the files for input to hardware
% -- file para.txt contains:
%    line 1: number of range cells
%    line 2: number of pulse in a batch (equals to dp_pts in this program)
%    line 3: extent of target in cells (n: integer); number of taps in delay
also equals n (pipeline design)
%    line 4: gain1, gain2, ..., gain n (integer)
%    line 4+n+1: phi0 (pulse 1),
%    line 4+n+2: phi1 (pulse 1),
%    line 4+n+targetExtent: phi-targetExtent (pulse 1),
%    line 4+n+targetExtent+1: phi0 (pulse 2),
%    line 4+n+targetExtent+2: phi1 (pulse 2),
%    line 4+n+2*targetExtent: phi-targetExtent (pulse 2),
%    ...
%    line 4+n+dp_pts*targetExtent: phi-targetExtent (pulse dp_pts)
%
% -- file raw.txt contains the instantaneous phases of simulated DFRM data
(quantized to 45deg step):
%    line 1: pulse 1 (integer)
%    line 2: pulse 2
%    ....
%    line dp_pts: pulse dp_pts
% 9 Jul 98
% SY Yeo
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear

set(0,'defaultAxesFontSize',8);

noplot = 0;
Ncontours = 20;

% Parameters
bw = 100e6;                                  %bandwidth of the chirp pulse
pwc = 1/(1.25*bw);                           %compressed pulse width
pw =0.5e-6;                                  %transmitted pulse width
prf = 2e3;                                   %PRF
pri = 1/prf;                                 %PRI
mu = 2*pi*bw/pw;                             %
fs = 1.25*bw;                                %sampling frequency
Ts = 1/fs;                                   %sampling period
snr = 0;                                     %SNR

% set-up grid
% x-axis(rg), 'y-axis(dp)
```

```
rg_pts = 200;                                %# of points on x-axis (range
cells)
dp_pts = 64;                                 %# of points on y-axis (doppler
cells)
pri_rg_map = zeros(dp_pts,rg_pts);
pri_rg_mapq = zeros(dp_pts,rg_pts);
pri_rg_map_shift = zeros(dp_pts,rg_pts);
pri_rg_map_shiftq = zeros(dp_pts,rg_pts);
pri_rg_phaseq = zeros(dp_pts,rg_pts);

% insert waveform into grid;
load -ascii sigpar1                          %load parameters from file sigpar1
sigpar = sigpar1';
sigpar([2 4],:) = sigpar([2 4],:)*prf/dp_pts;
[lxs,lys] = size(sigpar);
%t0 = Ts:Ts:pw;
t0 = 0:Ts:pw-Ts;
num_chirp_samples = length(t0);
if ((num_chirp_samples + lxs) > rg_pts)
   disp('Warning : Chirp is clipped - set grid size larger');
end

% open files for writing
f1 = fopen('para.txt','w');
fprintf(f1,'%d\r\n',num_chirp_samples);      % number of range cells
fprintf(f1,'%d\r\n',dp_pts);                 % number of doppler cells
fprintf(f1,'%d\r\n',lys);                    % target extent

% adjustment to correct multiplication factors for the amplitude (gain) value
for i = 1:lys
   switch sigpar(3,i)
   case {1}
      sigpar(3,i)=1; % no shift, multiplication by 1, hardware bit "00"
   case {2}
      sigpar(3,i)=2; % shift by 1, multiplication by 2, hardware bit "01"
   case {3}
      sigpar(3,i)=4; % shift by 2, multiplication by 4, hardware bit "10"
   case {4}
      sigpar(3,i)=8; % shift by 3, multiplication by 8, hardware bit "11"
   end
   fprintf(f1,'%d\r\n',sigpar(3,i));   % gain1, gain2, ..., gainN
end

nbitsph = 3;
nbitsdop = 5;
nbitsamp = 8;
b = 2*pi/(2^nbitsph);
a = 2*pi/(2^nbitsamp);
p = 2*pi/(2^nbitsdop);

for idx1 = 1:dp_pts     % Repeat for total number of pulses within batch
   %t1 = t0;
   %t1 = t0 + (idx1)*pri;
   t1 = t0 + (idx1-1)*pri;
   for idx = 1:lys
      %**** approximation used here, assume phase change due to doppler
within a chirp is constant
```

```
      %**** since the doppler is tens of hertz compared to the MHz chirp
bandwidth
      %oldphase = mu*t1.*t1/2 + 2*pi*sigpar(2,idx)*t1;
      oldphase = mu*t1.*t1/2 + 2*pi*sigpar(2,idx)*(idx1-1)*pri;
      oldphase = mod(oldphase,2*pi);

      % quantize the oldphase to 1 of 8 phases
      int_oldphase = round(oldphase/b);
      oldphaseq = b*int_oldphase;      % quantize the phase
      xc = exp(sqrt(-1)*oldphase);
      lx = (sigpar(1,idx)):(sigpar(1,idx))+length(xc)-1;
      pri_rg_map(idx1,lx) = xc+pri_rg_map(idx1,lx);
      pri_rg_phaseq(idx1,lx) = int_oldphase;

      xcq = exp(sqrt(-1)*oldphaseq);
      xcq = p*round(xcq/p);      % quantize the phase
      pri_rg_mapq(idx1,lx) = xcq+pri_rg_mapq(idx1,lx);
      % phase focusing
      %oscc = cos(2*pi*sigpar(4,idx)*t1);
      %oscs = sin(2*pi*sigpar(4,idx)*t1);
      %xI = real(xc).*oscc - imag(xc).*oscs;
      %xQ = imag(xc).*oscc + real(xc).*oscs;
      %dopphase = 2*pi*sigpar(4,idx)*t1;
      %newphase = oldphase + dopphase;
      %dopphase = 2*pi*sigpar(4,idx)*(idx1)*pri; % approximation used here
      dopphase = 2*pi*sigpar(4,idx)*(idx1-1)*pri; % approximation used here
      newphase = oldphase + dopphase*ones(size(oldphase));
      xI = cos(newphase);
      xQ = sin(newphase);
      x1 = sigpar(3,idx)*(xI+sqrt(-1)*xQ);
      pri_rg_map_shift(idx1,lx) = pri_rg_map_shift(idx1,lx) + x1;

      int_dopphaseq = round(dopphase/p);
      dopphaseq = int_dopphaseq*p;
      newphaseq = oldphaseq + dopphaseq;
      xI = cos(newphaseq);
      xQ = sin(newphaseq);
      xI = round(xI/a)*a;
      xQ = round(xQ/a)*a;
      x1 = sigpar(3,idx)*(xI+sqrt(-1)*xQ);
      pri_rg_map_shiftq(idx1,lx) = pri_rg_map_shiftq(idx1,lx) + x1;

      % store the dopphase value (ignore intrapulse phase change since it is
small)
      fprintf(f1,'%d\r\n',int_dopphaseq);
   end
end
fclose(f1);

noise = randn(size(pri_rg_map))*c_snr(snr); noise = 0;
pri_rg_map = pri_rg_map + noise;
pri_rg_map_shift = pri_rg_map_shift + noise;

%------
save pulse1 pri_rg_map_shiftq

%--------------
```

```
% Perform pulse compression
% (a) for the non-quantized phase case
disp('Creating reference waveform');
ph = (mu*t1.*t1/2);
crefc = exp(sqrt(-1)*ph);
cref = conj(fft(crefc,2*rg_pts-1));
save pc_ref cref
pc_ref_map = fft(pri_rg_map.',2*rg_pts-1).';
pc_ref_map_shift = fft(pri_rg_map_shift.',2*rg_pts-1).';

disp('Performing pulse compression');
pri_rg_map1 = zeros(size(pri_rg_map));
pri_rg_map2 = zeros(size(pri_rg_map));

%--- Compress the original signals
for idx = 1:dp_pts
   tmp = cref.*pc_ref_map(idx,:);
   tmp1 = fftshift(ifft(tmp));
   pri_rg_map1(idx,:) =  tmp1(rg_pts:end);
end

%--- Compress the doppler shifted signals

for idx = 1:dp_pts
   tmp = cref.*pc_ref_map_shift(idx,:);
   tmp1 = fftshift(ifft(tmp));
   pri_rg_map2(idx,:) =  tmp1(rg_pts:end);
end


% Compute the rg-dop map
disp('Plotting ... r-d map');

dp_rg_map = fft(pri_rg_map1);
dp_rg_map_shift = fft(pri_rg_map2);

[lx,ly] = size(dp_rg_map);
rax = 1:(length(ly));
dax = 0:(length(lx))-1;

dpy = abs(dp_rg_map);
dpy_shift = abs(dp_rg_map_shift);

if (noplot == 0)
   figure(1);

   subplot(2,1,1);
   h = contour(dpy,Ncontours); grid
   title('a. Original Rd-Dp Map');
   axis([1 62 0 64])
   xlabel('Down Range Cells');   ylabel('Cross Range Cells');
   subplot(2,1,2);
   h = contour(dpy_shift,Ncontours); grid
   axis([1 62 0 64])
   title('b. Amplitude and Doppler Modulated Rd-Dp Map');
   xlabel('Down Range Cells');   ylabel('Cross Range Cells');
```

```
    % Perform pulse compression
    % (b) for the quantized phase case
    disp('Performing pulse compression for quantized phase case');
    pc_ref_mapq = fft(pri_rg_mapq.',2*rg_pts-1).';
    pc_ref_map_shiftq = fft(pri_rg_map_shiftq.',2*rg_pts-1).';
    pri_rg_map3 = zeros(size(pri_rg_mapq));
    pri_rg_map4 = zeros(size(pri_rg_mapq));

    %--- Compress the original signals

    for idx = 1:dp_pts
       tmp = cref.*pc_ref_mapq(idx,:);
       tmp1 = fftshift(ifft(tmp));
       pri_rg_map3(idx,:) =  tmp1(rg_pts:end);
    end

    %--- Compress the doppler shifted signals

    for idx = 1:dp_pts
       tmp = cref.*pc_ref_map_shiftq(idx,:);
       tmp1 = fftshift(ifft(tmp));
       pri_rg_map4(idx,:) =  tmp1(rg_pts:end);
    end


    % Compute the rg-dop map
    disp('Plotting ... r-d map');

    dp_rg_mapq = fft(pri_rg_map3);
    dp_rg_map_shiftq = fft(pri_rg_map4);

    [lx,ly] = size(dp_rg_mapq);
    rax = 1:(length(ly));
    dax = 0:(length(lx))-1;

    dpyq = abs(dp_rg_mapq);
    dpy_shiftq = abs(dp_rg_map_shiftq);

    % -- Simulation of phase quantizing DRFM
    % Now convert amplitude to phase.
    % Convert phase to positive numbers between 0-360deg, so do not need to
handle
    % negative numbers in Altera.
    pri_rg_mapq_angle = mod(pri_rg_phaseq,2*pi);
    pri_rg_mapq_shift_angle = angle(pri_rg_map_shiftq);

    f2 = fopen('rawint.txt','w');
    [lx,ly] = size(pri_rg_mapq_angle);
    deltaDegrees = 2*pi/(2^nbitsdop);
    for i = 1:lx
       int_raw = round(pri_rg_mapq_angle(i,1:num_chirp_samples-
1)/deltaDegrees); % need to store in Visual basic text file format
       fprintf(f2,'%d,',int_raw);
       int_raw = round(pri_rg_mapq_angle(i,num_chirp_samples)/deltaDegrees);
       fprintf(f2,'%d\r\n',int_raw);
    end;
    fclose(f2);
```

```
    q = 2*pi/(2^nbitsph);
    pri_rg_mapq_drfm = exp(sqrt(-1)*(round(pri_rg_mapq_angle/q))*q);
    pri_rg_mapq_shift_drfm = exp(sqrt(-
1)*(round(pri_rg_mapq_shift_angle/q))*q);

    % Perform pulse compression
    % (c) for the quantized phase case with phase DFRM model
    disp('Performing pulse compression for quantized phase case (simulates
phase DFRM effects)');
    pc_ref_mapq_drfm = fft(pri_rg_mapq_drfm.',2*rg_pts-1).';
    pc_ref_mapq_shift_drfm = fft(pri_rg_mapq_shift_drfm.',2*rg_pts-1).';
    pri_rg_map5 = zeros(size(pri_rg_mapq_drfm));
    pri_rg_map6 = zeros(size(pri_rg_mapq_shift_drfm));

    %--- Compress the original signals
    for idx = 1:dp_pts
       tmp = cref.*pc_ref_mapq_drfm(idx,:);
       tmp1 = fftshift(ifft(tmp));
       pri_rg_map5(idx,:) =  tmp1(rg_pts:end);
    end

    %--- Compress the doppler shifted signals
    for idx = 1:dp_pts
       tmp = cref.*pc_ref_mapq_shift_drfm(idx,:);
       tmp1 = fftshift(ifft(tmp));
       pri_rg_map6(idx,:) =  tmp1(rg_pts:end);
    end

    % Compute the rg-dop map
    disp('Plotting ... r-d map');

    dp_rg_mapq_drfm = fft(pri_rg_map5);
    dp_rg_map_shiftq_drfm = fft(pri_rg_map6);

    [lx,ly] = size(dp_rg_mapq_drfm);
    rax = 1:(length(ly));
    dax = 0:(length(lx));
    %dax = 0:(length(lx))-1;

    dpyq_drfm = abs(dp_rg_mapq_drfm);
    dpyq_shift_drfm = abs(dp_rg_map_shiftq);
    save plot dpyq dpyq_shift_drfm

end

figure(1); print -dtiff simhost1
```

simhwchkv1.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% simhwchkv1.m
%
% Modified version of simhwchk.m
% Modified by: Stig Ekestorm, Aug-99
%
% simhwchk.m
% Purpose: This program performs a architectural true simulation of the
% Digital Image Synthesizer hardware
% Created by: SY Yeo
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear

set(0,'defaultAxesFontSize',8);

noPlot = 0;
Ncontours = 20;

depthLUT = 32;
widthLUTFile = 2; % in units of number of hex digits
widthLUT = 8;  % n bits
ndopbits = 5;
rg_pts = 200;

%*****************************
% Read from data files
%*****************************
% Read from para.dat
fid = fopen('para.txt','r'); %opens para.txt to be read
tmp = fscanf(fid,'%d'); %reads in the values
nRangeCell = tmp(1); %1st value: 62, represents the number of range cells
nDopplerCell = tmp(2); %2nd value: 64, represents the number of radar pulses
targetExtent = tmp(3); %3rd value: 3, represents the radial length of the
target expressed in number of range cells
gain = tmp(4:4+targetExtent-1); %4th to 6th values: 1,2,4 - the gain value
for each tap
gainRev = fliplr(gain);
tmp1 = tmp(4+targetExtent:end); % 7th to last value: the phase increment
values for each tap
%phi = reshape(tmp1,targetExtent,nDopplerCell+(targetExtent-2));
phi = reshape(tmp1,targetExtent,nDopplerCell); %3x64 matrix with zeros in the
1st column
fclose(fid);

% Read from rawint.dat
raw = zeros(nDopplerCell,nRangeCell); %create a 64x62 matrix, initialized to
zeros
fid = fopen('rawint.txt','r'); %open rawint.txt to be read
for j = 1:nDopplerCell
   for k = 1:nRangeCell-1
      raw(j,k) = fscanf(fid,'%d',1);
      comma = fscanf(fid,'%c',1);
   end
```

```
   raw(j,nRangeCell) = fscanf(fid,'%d',1);
end
fclose(fid);
[row,col] = size(raw);
raw = [raw,zeros(row,targetExtent-1)]; %raw: 64x64 matrix, last 2 columns
with zeros
%raw = [raw,zeros(row,targetExtent)]; %raw: 64x65 matrix, last 3 columns with
zeros

% Read from the LUT files.
load -ascii cosine.txt  % variable is cosine
load -ascii sine.txt % variable is sin

%*****************************
% Pulse-Pulse Processing
%*****************************
delayLine = zeros(targetExtent+1,1);
gainRev = fliplr(gain);
phiRev = zeros(targetExtent,1);

% Intermediate node variables
phaseAdderOut = zeros(targetExtent,1);
lutOut = zeros(targetExtent,1);
finalAdderOut = zeros(nDopplerCell,nRangeCell + (targetExtent-1),1);
%finalAdderOut = zeros(nDopplerCell,nRangeCell + targetExtent,1);

f1 = fopen('checkv1.txt','w');

for batchCnt = 1:nDopplerCell

   disp(['Processing Pulse 'num2str(batchCnt)]);

   for intraPulseCnt = 1:(nRangeCell + (targetExtent-1)) % clock cycle
   %for intraPulseCnt = 1:(nRangeCell + targetExtent) % clock cycle

      delayLine(2:targetExtent+1) = delayLine(1:targetExtent);
      delayLine(1) = raw(batchCnt,intraPulseCnt);

      % --- This part simulates the intra pulse processing in hardware

      % Phase addition (add doppler offset to DFRM phase)
      phaseAdderOut = delayLine(1:targetExtent) + phi(:,batchCnt);

      % Phase-amplitude look-up (to obtain complex time signal)
      tmp = mod(phaseAdderOut,depthLUT) + 1;
      %tmp = phaseAdderOut + 1;
      lutOut = cosine(tmp) + sqrt(-1)*sine(tmp);

      %To correct processing flow (ref. Yeo's Thesis, p.35)
      %intraPulseCnt = 1:((nRangeCell) + (targetExtent-1))
      %correction at start-up ("initialize one tap after another")
      if intraPulseCnt<targetExtent,
         test=1:1:(targetExtent-1);
         idx=test(:,intraPulseCnt);
         for idx2=intraPulseCnt:(targetExtent-1),
            lutOut((idx2+1),:)=0;
         end
```

```
      end
      %correction at the end ("shutting down the taps one tap after another")
      if intraPulseCnt>nRangeCell,
         for idx2=1:(intraPulseCnt-nRangeCell),
            lutOut((idx2),:)=0;
         end
      end

      % Gain modulation
      lutOut1 = gainRev.*lutOut;

      % Final accumulation to form extended target
      finalAdderOut(batchCnt,intraPulseCnt) = sum(lutOut1);

      fprintf(f1,'%s:%d\r\n','clock cycle',intraPulseCnt);
      fprintf(f1,'DL:%d ',delayLine); fprintf(f1,'\t');
      fprintf(f1,'phi:%d ',phi(:,batchCnt)); fprintf(f1,'\t');
      fprintf(f1,'modph:%d ',tmp);
      fprintf(f1,'\r\n');
      fprintf(f1,'rlutOut:%5.2f ',real(lutOut)); fprintf(f1,'\t');
      fprintf(f1,'ilutOut:%5.2f ',imag(lutOut)); fprintf(f1,'\t');
      fprintf(f1,'\r\n');
      fprintf(f1,'rsum:%5.2f ',real(sum(lutOut1))); fprintf(f1,'\t');
      fprintf(f1,'isum:%5.2f ',real(sum(lutOut1))); fprintf(f1,'\t');
      fprintf(f1,'\r\n');
      fprintf(f1,'\r\n');
   end % intraPulseCnt
end % batchCnt
fclose(f1);

%*****************************
% Pulse Compression
%*****************************
%--- Compress the doppler shifted signals
load pc_ref
priRgMapShift = zeros(nDopplerCell,rg_pts);
tic
pcRefMapShift = fft(finalAdderOut.',2*rg_pts-1).';
for idx = 1:nDopplerCell
   tmp = cref.*pcRefMapShift(idx,:);
   tmp1 = fftshift(ifft(tmp));
   priRgMapShift(idx,1:end-targetExtent+1) =  tmp1(rg_pts+targetExtent-
1:end);
end
dpRgMapShiftMOD = abs(fft(priRgMapShift));
%dpRgMapShift = abs(fft(priRgMapShift));
toc

save plotMOD dpRgMapShiftMOD

save fAddOut finalAdderOut

%*****************************
% Display
%*****************************
if (noPlot == 0)
   figure(2);
```

```
   load plot.mat
   subplot(2,1,1);
   h = contour(dpyq,Ncontours); grid; axis([1 62 0 64])
   %h = contour(dpyq_shift_drfm,Ncontours); grid; axis([0 20 0 32])
   title('a. Amplitude and Doppler Modulated Rd-Dp Map (unmodulated / MATLAB)
');
   xlabel('Down Range Cells');   ylabel('Cross Range Cells');
   axis([1 62 0 64])
   subplot(2,1,2);
   h = contour(dpRgMapShiftMOD,Ncontours); grid; axis([1 62 0 64])
   %h = contour(dpRgMapShift,Ncontours); grid; axis([1 20 0 32])
   title('b. Amplitude and Doppler Modulated Rd-Dp Map (Bit-True, modulated /
MATLAB) ');
   xlabel('Down Range Cells');   ylabel('Cross Range Cells');
   axis([1 62 0 64])

end
```

plothwv1.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% plothwv1.m
%
% Modified version of plothwv0.m
% Modified by Stig Ekestorm, Aug -99
% This version processes the output from the hardware
% Works in concert with mathostv0.m and simhwchk.m
% Created by SY Yeo,  11 Aug 1998
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear

set(0,'defaultAxesFontSize',7)

noplot = 0;

rg_pts = 200;

%to load data from hardware output files
load -ascii imagei.txt
load -ascii imageq.txt

fid = fopen('para.txt','r');
tmp = fscanf(fid,'%d');
nRangeCell = tmp(1);
nDopplerCell = tmp(2);
targetExtent = tmp(3);
fclose(fid);

%for getting the data form hardware in the right format
image = imagei - j*imageq;
image = reshape(image,nRangeCell+targetExtent,nDopplerCell); %for FPGA 3-tap
simulation
%image = reshape(image,nRangeCell+(targetExtent-1),nDopplerCell); %for ASIC
simulation
image = image';

load fAddOut

if (noplot == 0)
   %****************************
   % Pulse Compression
   %****************************
   %--- Compress the doppler shifted signals
   figure(3);
   orient tall

   load plot.mat
   load plotMOD.mat

   Ncontours = 9;
   subplot(2,1,1);
   %h = contour(dpyq_shift_drfm,Ncontours); grid, axis([1 62 0 64])
   h = contour(dpRgMapShiftMOD,Ncontours); grid; axis([1 62 0 64])
```

```
   title('a. Amplitude and Doppler Modulated Rd-Dp Map (Bit and Architecture-
True / MATLAB)');
   xlabel('Down Range Cells');   ylabel('Cross Range Cells');

   %to post-process data from hardware
   load pc_ref
   priRgMapShift = zeros(nDopplerCell.',rg_pts);
   tic
   pcRefMapShift = fft(image.',2*rg_pts-1).';
   for idx = 1:nDopplerCell
      tmp = cref.*pcRefMapShift(idx,:);
      tmp1 = fftshift(ifft(tmp));
      priRgMapShift(idx,1:end-targetExtent+1) =  tmp1(rg_pts+targetExtent-
1:end);
   end
   dpRgMapShift = abs(fft(priRgMapShift));
   toc

   subplot(2,1,2);
   h = contour(dpRgMapShift,Ncontours);
   grid, axis([1 62 0 64])
   title('b. Amplitude and Doppler Modulated Rd-Dp Map (from HARDWARE
output)');
   xlabel('Down Range Cells');   ylabel('Cross Range Cells');
end

figure(3)
print -dtiff hwres

%figure(5)
figure(4)
subplot(3,1,1);
h = mesh(dpRgMapShiftMOD); grid;
%h = mesh(dpyq_shift_drfm); grid;
title('a. Amplitude/Doppler Modulated Rd-Dp Map (Bit-True, modulated /
MATLAB)');
xlabel('Down Range Cells');   ylabel('Cross Range Cells'); grid
subplot(3,1,2);
%to plot hardware output
h = mesh(dpRgMapShift); grid;
title('b. Amplitude/Doppler Modulated Rd-Dp Map (HARDWARE output)');
xlabel('Down Range Cells');   ylabel('Cross Range Cells'); grid
subplot(3,1,3);
%to plot difference between Matlab simulation and hardware output
%h = mesh(dpyq_shift_drfm/max(max(dpyq_shift_drfm))-
dpRgMapShift/max(max(dpRgMapShift))); grid;
h = mesh(dpRgMapShiftMOD-dpRgMapShift); grid; %plot the real difference, Stig
Aug-99
%h = mesh(dpyq_shift_drfm-dpRgMapShift); grid; %plot the real difference,
Stig Aug-99
title('c. Difference');
xlabel('Down Range Cells');   ylabel('Cross Range Cells'); grid
print -dtiff diffplot

%for comparison 5 Oct -99, Stig Ekestorm
%figure(5)
%h = mesh(dpRgMapShiftMOD); grid;
```

```
%title('a. Amplitude/Doppler Modulated Rd-Dp Map (Bit-True, modulated /
MATLAB)');
%xlabel('Down Range Cells');  ylabel('Cross Range Cells'); grid

%for comparison 5 Oct -99, Stig Ekestorm
%figure(6)
%h = mesh(dpyq_shift_drfm); grid;
%title('a. Amplitude/Doppler Modulated Rd-Dp Map (Bit-True, modulated /
MATLAB)');
%xlabel('Down Range Cells');  ylabel('Cross Range Cells'); grid
```

Version 2

runDISv2.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% runDISv2.m
% This script file will help you to run the Digital Image Syntesizer (DIS)
% This is a modified version that is able to handle different target extents
% (that is, how many taps the user would like to use that will represent the
% radial length of the target, seen from the ISAR)
% The user can also specify some necessary input parameters
% Created by:
%   MAJ Stig Ekestorm, Nov -99
%   Naval Postgraduate School
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% set path
% cd c:\temasek\denise\thesis\final_design\vbfiles

% clear the workspace
clear

% declare global variables, used in outer m-files and functions
global dp_pts
global rg_pts
global hda

% interactive - use of a dialog box to get inputs parameters from user
title='User Specified Parameters - Matlab DIS Simulation';
prompt={'Number of Doppler cells in the ISAR',...
        'Number of Range gates in the plots',...
        'Hardware Data available for comparison [1 for yes]'};
default={'64','200','0'};
response=inputdlg(prompt, title, 1, default);
fields={'dp_pts','rg_pts','hda'}; % number of Doppler cells, hardware data
available
input=cell2struct(response,fields,1);
% convert cell structure created by dialog box back to numbers
dp_pts=str2num(input.dp_pts);
rg_pts=str2num(input.rg_pts);
hda=str2num(input.hda);

% run the graphical user interface (GUI) to specify target parameters
guiv2
disp('Enter the values in the Grapical User Interface')
disp('Press any key to continue')
pause

% pre-process signal parameters, simulate ISAR
mathostv2

% simulate the DIS in Matlab
% This simulation does "parallel processing" and then "serial summation",
including:
% - correction at start-up ("initializing outputs from the taps, one tap
after another")
```

```
% - correction at the end ("shutting down the taps, one tap after another")
simhwchkv2

% plot results for visual comparison
plothwv2

% end of file
```

---

guiv2.m

```
function [dat] = guiv2(action);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% guiv2.m
% Get inputs from screen
% MAJ Stig Ekestorm, Sep -99
% Modified version of guiv2.m by SY YEO, 30 Jan 98
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


global hf
global h1
global h2
global data
global loc
global patchsize
global txtloc
global count
global ph
global dp_pts

if nargin<1,
    action='start';
end;

if strcmp(action,'start'),

   % initialize the figure

   set(0,'DefaultAxesFontSize',6);

   hf = figure(1); clf
   set(hf,'NextPlot','add');

   set(hf, ...
      'NumberTitle','off', ...
      'Name','Naval PostGraduate School',...
      'backingstore','off',...
      'Units','normalized');


   %rg_pts = 15;
   rg_pts = 62;
   %dp_pts = 64;
   data = [];  loc = [];
   count = 0;
   ph = [];

   h1 = axes('Position',[0 0 1 1],'Visible','off');
   h2 = axes('Position',[0.1 0.1 0.6 0.8]);

   set(hf,'currentaxes',h2);

   xa = 1:rg_pts;
```

```
ya = 0:(dp_pts-1);

xtick = 0:1:rg_pts;
set(gca,'XTickMode','manual');
set(gca,'XLimMode','manual');
set(gca,'XLim',[1 rg_pts]);
set(gca,'XTick',xtick);
set(gca,'XGrid','on');
set(gca,'GridLineStyle','-');

set(gca,'YTickMode','manual');
set(gca,'YLimMode','manual');
set(gca,'YLim',[0 dp_pts-1]);
ytick = 0:1:dp_pts;
set(gca,'YTick',ytick);
set(gca,'YGrid','on');
set(gca,'GridLineStyle','-');

xh = xlabel('Range Cell'); set(xh,'FontSize',8); clear xh
yh = ylabel('Doppler'); set(yh,'FontSize',8); clear yh
ht = title('Range-Doppler-Amplitude Map Entry Program');
set(ht,'FontSize',10,'Color',[0 0 1]);

a = uicontrol('Units','normalized', ...
   'BackgroundColor',[.9 .9 .9], ...
   'Position',[0.72 0.80 0.15 0.04], ...
   'Style','text', ...
   'FontSize',6,...
   'String','Range Cell', ...
   'Tag','aText');

b = uicontrol('Units','normalized', ...
   'BackgroundColor',[.9 .9 .9], ...
   'Position',[0.72 0.75 0.15 0.04], ...
   'Style','text', ...
   'FontSize',6,...
   'String','Doppler Cell', ...
   'Tag','bText');

c = uicontrol('Units','normalized', ...
   'BackgroundColor',[.9 .9 .9], ...
   'Position',[0.72 0.65 0.15 0.04], ...
   'Style','text', ...
   'FontSize',6,...
   'String','Amplitude', ...
   'Tag','cText');

c11 = uicontrol('Units','normalized', ...
   'BackgroundColor',[.9 .9 .9], ...
   'Position',[0.72 0.60 0.15 0.04], ...
   'Style','slider','min',0,'max',4,...
   'SliderStep',[0.25 0.5],...
   'Callback','guiv2(''update1'')');

d = uicontrol('Units','normalized', ...
   'BackgroundColor',[.9 .9 .9], ...
   'Position',[0.72 0.50 .15 0.04], ...
```

```
    'Style','text', ...
    'FontSize',6,...
    'String','Doppler shift');

d11 = uicontrol('Units','normalized', ...
    'BackgroundColor',[.9 .9 .9], ...
    'Position',[0.72 0.45 0.15 0.04], ...
    'Style','slider','Min',-10,'Max',10,...
    'SliderStep',[0.05 0.1],...
    'Callback','guiv2(''update1'')');

a1 = uicontrol('Units','normalized', ...
    'BackgroundColor',[1 1 1], ...
    'Position',[0.9 0.80 0.05 0.04], ...
    'Style','text', ...
    'FontSize',6,...
    'String','', ...
    'Tag','a1Text');

set(gcf,'currentaxes',h1);
b1 = uicontrol('Units','normalized', ...
    'BackgroundColor',[1 1 1], ...
    'Position',[0.9 0.75 0.05 0.04], ...
    'Style','text', ...
    'FontSize',6,...
    'String','', ...
    'Tag','a2Text2');

set(gcf,'currentaxes',h1);
c1 = uicontrol('Units','normalized', ...
    'BackgroundColor',[1 1 1], ...
    'Position',[0.9 0.65 0.05 0.04], ...
    'Style','text', ...
    'FontSize',6,...
    'Callback','guiv2(''update'')',...
    'String','');

d1 = uicontrol('Units','normalized', ...
    'BackgroundColor',[1 1 1], ...
    'Position',[0.9 0.50 0.05 0.04], ...
    'Style','text', ...
    'FontSize',6,...
    'Callback','guiv2(''update'')',...
    'String','');

q1 = uicontrol('Units','normalized', ...
    'BackgroundColor','Yellow', ...
    'Position',[0.9 0.10 0.05 0.04], ...
    'Style','pushbutton', ...
    'FontSize',8,...
    'String','SAVE', ...
    'Callback','guiv2(''savequit'')');

q2 = uicontrol('Units','normalized', ...
    'BackgroundColor','Yellow', ...
    'Position',[0.78 0.1 0.1 0.04], ...
    'Style','pushbutton', ...
```

```
        'FontSize',8,...
        'String','CLEAR', ...
        'Callback','guiv2(''start'')');

    txtloc = [a a1 b b1 c c1 c11 d d1 d11];
    % Assign action when mouse button is pressed
    set(h2,'ButtonDownFcn','guiv2(''down'')');

elseif strcmp(action,'down'),
    % Obtain coordinates of mouse click location in axes units

    set(hf,'currentaxes',h2);
    pt=get(h2,'currentpoint');

    x=pt(1,1); xf = floor(x);
    y=pt(1,2); yf = floor(y);
    [r,c] = size(data);

    set(txtloc(7),'Value',0);
    set(txtloc(9),'Value',0);

    tmp = [x y 1 0];
    loc = [loc tmp];
    tmp = [xf yf 1 0];
    data = [data;tmp];
    [r,c] = size(data);
    ypos = [yf yf+1 yf+1 yf];
    xpos = [xf xf xf+1 xf+1];

    count = count + 1;
    %disp(count);
    txt = ['Tag',num2str(count)];
    ptr = patch(xpos,ypos,[1 1 1]*0.9);
    %disp(ptr);
    set(ptr,'ButtonDownFcn',[ ...
          'guiv2(''update'')']);
    set(ptr,'Tag',txt);
    set(ptr,'UserData',[xf yf 1 0]);
    ph = ptr;
    set(txtloc(2),'String',xf);
    set(txtloc(4),'String',yf);
    set(txtloc(6),'String',1);
    set(txtloc(9),'String',0);

elseif strcmp(action,'update'),
    % Determine the patch that is selected
    ph = gcbo;
    %set(ph,'Selected','on');
    % Retrieve the values for that patch and display it
    % txtloc = [a a1 b b1 c c1 c11 d d1 d11];
    % txtloc 2: Range cell
    % txtloc 4: Doppler cell
    % txtloc 6: Amplitude  txtloc 7: Slider bar
    % txtloc 9: Doppler offset  txtloc 9: Slider bar
    ud = get(ph,'UserData');
    set(txtloc(2),'String',ud(1));
    set(txtloc(4),'String',ud(2));
```

```
      set(txtloc(6),'String',ud(3));
      set(txtloc(9),'String',ud(4));
      set(txtloc(7),'Value',ud(3));
      set(txtloc(10),'Value',ud(4));

   elseif strcmp(action,'update1'),
      if (~isempty(ph))
         ph1 = gcbo;
         if ((ph1 == txtloc(7)) | (ph1 == txtloc(10)))
            ud = get(ph,'UserData');
            xf = ud(1);   yf = ud(2);
            ypos = [yf yf+1 yf+1 yf];
            xpos = [xf xf xf+1 xf+1];
            set(ph,'Selected','off');
            % Update the amplitude/doppler values
            if (ph1 == txtloc(7))
               tmp1 = get(txtloc(7),'Value');
               tmp1 = round(tmp1)
               set(txtloc(6),'String',tmp1);
               set(txtloc(7),'Value',tmp1);
               if (tmp1 < 1),
                  set(txtloc(7),'Value',1);
                  set(txtloc(6),'String','1');
                  tmp1 = 1;
               end
               col = [1 1 1]*(1-tmp1/10);
               set(ph,'FaceColor',col);
               set(ph,'UserData',[ud(1) ud(2) tmp1 ud(4)]);
            end
            if (ph1 == txtloc(10))
               tmp2 = round(get(txtloc(10),'Value'));
               set(txtloc(9),'String',tmp2);
               set(txtloc(10),'Value',tmp2);
               set(ph,'UserData',[ud(1) ud(2) ud(3) tmp2]);
            end
            %disp('HHH');
            %disp(get(ph,'Tag'))
            %disp(get(ph,'UserData'))
         end
      end

elseif strcmp(action,'savequit'),
   dat = [];
   for i = 1:count
      tt = findobj('Tag',['Tag' num2str(i)]);
      tmp = get(tt,'UserData')
      dat = [dat;tmp];
      fprintf('count = %d, Tag = %s  ',count,get(tt,'Tag'));
      disp(tmp);
   end
   save -ascii sigpar1 dat
   close gcbf
end
```

mathostv2.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% mathostv2.m
% MAJ Stig Ekestorm, Sep -99
% Modified version of mathostv0.m by SY YEO, 9 Jul 98
%
% Generate pri_dp map and range-doppler map
% - generates the files for input to hardware
% -- file para.txt contains:
%    line 1: number of range cells
%    line 2: number of pulse in a batch (equals to dp_pts in this program)
%    line 3: extent of target in cells (n: integer); number of taps in delay
also equals n (pipeline design)
%    line 4: gain1, gain2, ..., gain n (integer)
%    line 4+n+1: phi0 (pulse 1),
%    line 4+n+2: phi1 (pulse 1),
%    line 4+n+targetExtent: phi-targetExtent (pulse 1),
%    line 4+n+targetExtent+1: phi0 (pulse 2),
%    line 4+n+targetExtent+2: phi1 (pulse 2),
%    line 4+n+2*targetExtent: phi-targetExtent (pulse 2),
%    ...
%    line 4+n+dp_pts*targetExtent: phi-targetExtent (pulse dp_pts)
%
% -- file raw.txt contains the instantaneous phases of simulated DFRM data
(quantized to 45deg step):
%    line 1: pulse 1 (integer)
%    line 2: pulse 2
%    ....
%    line dp_pts: pulse dp_pts
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear

global dp_pts
global rg_pts
global doppler_inc

set(0,'defaultAxesFontSize',8);

noplot = 0;
Ncontours = 20;

% Parameters
bw = 100e6;
pwc = 1/(1.25*bw);   % compressed pulsewidth
pw =0.5e-6;
prf = 2e3;  pri = 1/prf;
mu = 2*pi*bw/pw;
fs = 1.25*bw;  Ts = 1/fs;
snr = 0;

% set-up grid
% x-axis(rg), y-axis(dp)
%rg_pts = 200;
%dp_pts = 64;
```

```
pri_rg_map = zeros(dp_pts,rg_pts);
pri_rg_mapq = zeros(dp_pts,rg_pts);
pri_rg_map_shift = zeros(dp_pts,rg_pts);
pri_rg_map_shiftq = zeros(dp_pts,rg_pts);
pri_rg_phaseq = zeros(dp_pts,rg_pts);

% insert waveform into grid;
load -ascii sigpar1
sigpar = sigpar1';
doppler_inc = sigpar(4,:);
sigpar([2 4],:) = sigpar([2 4],:)*prf/dp_pts;
[lxs,lys] = size(sigpar);
%t0 = Ts:Ts:pw;
t0 = 0:Ts:pw-Ts;


num_chirp_samples = length(t0); if ((num_chirp_samples + lxs) > rg_pts)
disp('Warning : Chirp is clipped - set grid size larger'); end

% open files for writing
f1 = fopen('para.txt','w');
fprintf(f1,'%d\r\n',num_chirp_samples);      % number of range cells
fprintf(f1,'%d\r\n',dp_pts);                 % number of doppler cells
fprintf(f1,'%d\r\n',lys);                    % target extent

% adjustment to correct multiplication factors for the amplitude (gain) value
for i = 1:lys
   switch sigpar(3,i)
   case {1}
      sigpar(3,i)=1; % no shift, multiplication by 1, hardware bit "00"
   case {2}
      sigpar(3,i)=2; % shift by 1, multiplication by 2, hardware bit "01"
   case {3}
      sigpar(3,i)=4; % shift by 2, multiplication by 4, hardware bit "10"
   case {4}
      sigpar(3,i)=8; % shift by 3, multiplication by 8, hardware bit "11"
   end
   fprintf(f1,'%d\r\n',sigpar(3,i));   % gain1, gain2, ..., gainN
end

nbitsph = 3;
nbitsdop = 5;
nbitsamp = 8;
b = 2*pi/(2^nbitsph);
a = 2*pi/(2^nbitsamp);
p = 2*pi/(2^nbitsdop);
%for the reduced 2-Tap T-Spice simulation, Nov -99
%p = 2*pi/(10);


for idx1 = 1:dp_pts     % Repeat for total number of pulses within batch
   %t1 = t0 + (idx1)*pri;
   t1 = t0 + (idx1-1)*pri;
   for idx = 1:lys
      %**** approximation used here, assume phase change due to doppler
within a chirp is constant
```

```
      %**** since the doppler is tens of hertz compared to the MHz chirp
 bandwidth
      %oldphase = mu*t1.*t1/2 + 2*pi*sigpar(2,idx)*t1;
      oldphase = mu*t1.*t1/2 + 2*pi*sigpar(2,idx)*(idx1-1)*pri;
      oldphase = mod(oldphase,2*pi);

      % quantize the oldphase to 1 of 8 phases
      int_oldphase = round(oldphase/b);
      oldphaseq = b*int_oldphase;      % quantize the phase
      xc = exp(sqrt(-1)*oldphase);
      lx = (sigpar(1,idx)):(sigpar(1,idx))+length(xc)-1;
      pri_rg_map(idx1,lx) = xc+pri_rg_map(idx1,lx);
      pri_rg_phaseq(idx1,lx) = int_oldphase;

      xcq = exp(sqrt(-1)*oldphaseq);
      xcq = p*round(xcq/p);      % quantize the phase
      pri_rg_mapq(idx1,lx) = xcq+pri_rg_mapq(idx1,lx);
      % phase focusing
      %dopphase = 2*pi*sigpar(4,idx)*(idx1)*pri; % approximation used here
      dopphase = 2*pi*sigpar(4,idx)*(idx1-1)*pri; % approximation used here
      newphase = oldphase + dopphase*ones(size(oldphase));
      xI = cos(newphase);
      xQ = sin(newphase);
      x1 = sigpar(3,idx)*(xI+sqrt(-1)*xQ);
      pri_rg_map_shift(idx1,lx) = pri_rg_map_shift(idx1,lx) + x1;

      int_dopphaseq = round(dopphase/p);
      dopphaseq = int_dopphaseq*p;
      newphaseq = oldphaseq + dopphaseq;
      xI = cos(newphaseq);
      xQ = sin(newphaseq);
      xI = round(xI/a)*a;
      xQ = round(xQ/a)*a;
      x1 = sigpar(3,idx)*(xI+sqrt(-1)*xQ);
      pri_rg_map_shiftq(idx1,lx) = pri_rg_map_shiftq(idx1,lx) + x1;

      % store the dopphase value (ignore intrapulse phase change since it is
small)
      fprintf(f1,'%d\r\n',int_dopphaseq);
   end
end
fclose(f1);

noise = randn(size(pri_rg_map))*c_snr(snr); noise = 0;
pri_rg_map = pri_rg_map + noise;
pri_rg_map_shift = pri_rg_map_shift + noise;

%------
save pulse1 pri_rg_map_shiftq

%--------------
% Perform pulse compression
% (a) for the non-quantized phase case
disp('Creating reference waveform');
ph = (mu*t1.*t1/2);
crefc = exp(sqrt(-1)*ph);
cref = conj(fft(crefc,2*rg_pts-1));
```

```
save pc_ref cref
pc_ref_map = fft(pri_rg_map.',2*rg_pts-1).';
pc_ref_map_shift = fft(pri_rg_map_shift.',2*rg_pts-1).';

disp('Performing pulse compression');
pri_rg_map1 = zeros(size(pri_rg_map));
pri_rg_map2 = zeros(size(pri_rg_map));

%--- Compress the original signals
for idx = 1:dp_pts
   tmp = cref.*pc_ref_map(idx,:);
   tmp1 = fftshift(ifft(tmp));
   pri_rg_map1(idx,:) =  tmp1(rg_pts:end);
end

%--- Compress the doppler shifted signals

for idx = 1:dp_pts
   tmp = cref.*pc_ref_map_shift(idx,:);
   tmp1 = fftshift(ifft(tmp));
   pri_rg_map2(idx,:) =  tmp1(rg_pts:end);
end

% Compute the rg-dop map
disp('Plotting ... r-d map');

dp_rg_map = fft(pri_rg_map1);
dp_rg_map_shift = fft(pri_rg_map2);

[lx,ly] = size(dp_rg_map);
rax = 1:(length(ly));
dax = 0:(length(lx))-1;

dpy = abs(dp_rg_map);
dpy_shift = abs(dp_rg_map_shift);

if (noplot == 0)
   figure(1);

   subplot(2,1,1);
   h = contour(dpy,Ncontours); grid
   title('a. Original Rd-Dp Map');
   axis([1 62 0 dp_pts])
   xlabel('Down Range Cells');   ylabel('Cross Range Cells');
   subplot(2,1,2);
   h = contour(dpy_shift,Ncontours); grid
   axis([1 62 0 dp_pts])
   title('b. Amplitude and Doppler Modulated Rd-Dp Map');
   xlabel('Down Range Cells');   ylabel('Cross Range Cells');

   % Perform pulse compression
   % (b) for the quantized phase case
   disp('Performing pulse compression for quantized phase case');
   pc_ref_mapq = fft(pri_rg_mapq.',2*rg_pts-1).';
   pc_ref_map_shiftq = fft(pri_rg_map_shiftq.',2*rg_pts-1).';
   pri_rg_map3 = zeros(size(pri_rg_mapq));
   pri_rg_map4 = zeros(size(pri_rg_mapq));
```

```
   %--- Compress the original signals

   for idx = 1:dp_pts
      tmp = cref.*pc_ref_mapq(idx,:);
      tmp1 = fftshift(ifft(tmp));
      pri_rg_map3(idx,:) =  tmp1(rg_pts:end);
   end

   %--- Compress the doppler shifted signals

   for idx = 1:dp_pts
      tmp = cref.*pc_ref_map_shiftq(idx,:);
      tmp1 = fftshift(ifft(tmp));
      pri_rg_map4(idx,:) =  tmp1(rg_pts:end);
   end

   % Compute the rg-dop map
   disp('Plotting ... r-d map');

   dp_rg_mapq = fft(pri_rg_map3);
   dp_rg_map_shiftq = fft(pri_rg_map4);

   [lx,ly] = size(dp_rg_mapq);
   rax = 1:(length(ly));
   dax = 0:(length(lx))-1;

   dpyq = abs(dp_rg_mapq);
   dpy_shiftq = abs(dp_rg_map_shiftq);

   % -- Simulation of phase quantizing DRFM
   % Now convert amplitude to phase.
   % Convert phase to positive numbers between 0-360deg, so do not need to
handle
   % negative numbers in Altera.
   pri_rg_mapq_angle = mod(pri_rg_phaseq,2*pi);
   pri_rg_mapq_shift_angle = angle(pri_rg_map_shiftq);

   f2 = fopen('rawint.txt','w');
   [lx,ly] = size(pri_rg_mapq_angle);
   deltaDegrees = 2*pi/(2^nbitsdop);
   for i = 1:lx
      int_raw = round(pri_rg_mapq_angle(i,1:num_chirp_samples-
1)/deltaDegrees); % need to store in Visual basic text file format
      fprintf(f2,'%d,',int_raw);
      int_raw = round(pri_rg_mapq_angle(i,num_chirp_samples)/deltaDegrees);
      fprintf(f2,'%d\r\n',int_raw);
   end;
   fclose(f2);
   q = 2*pi/(2^nbitsph);
   pri_rg_mapq_drfm = exp(sqrt(-1)*(round(pri_rg_mapq_angle/q))*q);
   pri_rg_mapq_shift_drfm = exp(sqrt(-
1)*(round(pri_rg_mapq_shift_angle/q))*q);

   % Perform pulse compression
   % (c) for the quantized phase case with phase DFRM model
```

```
   disp('Performing pulse compression for quantized phase case (simulates
phase DFRM effects)');
   pc_ref_mapq_drfm = fft(pri_rg_mapq_drfm.',2*rg_pts-1).';
   pc_ref_mapq_shift_drfm = fft(pri_rg_mapq_shift_drfm.',2*rg_pts-1).';
   pri_rg_map5 = zeros(size(pri_rg_mapq_drfm));
   pri_rg_map6 = zeros(size(pri_rg_mapq_shift_drfm));

   %--- Compress the original signals

   for idx = 1:dp_pts
      tmp = cref.*pc_ref_mapq_drfm(idx,:);
      tmp1 = fftshift(ifft(tmp));
      pri_rg_map5(idx,:) =  tmp1(rg_pts:end);
   end

   %--- Compress the doppler shifted signals

   for idx = 1:dp_pts
      tmp = cref.*pc_ref_mapq_shift_drfm(idx,:);
      tmp1 = fftshift(ifft(tmp));
      pri_rg_map6(idx,:) =  tmp1(rg_pts:end);
   end

   % Compute the rg-dop map
   disp('Plotting ... r-d map');

   dp_rg_mapq_drfm = fft(pri_rg_map5);
   dp_rg_map_shiftq_drfm = fft(pri_rg_map6);

   [lx,ly] = size(dp_rg_mapq_drfm);
   rax = 1:(length(ly));
   dax = 0:(length(lx));
   %dax = 0:(length(lx))-1;

   dpyq_drfm = abs(dp_rg_mapq_drfm);
   dpyq_shift_drfm = abs(dp_rg_map_shiftq);

   save plot dpyq dpyq_shift_drfm

end

figure(1); print -dtiff simhost1
```

simhwchkv2.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% simhwchkv2.m
% MAJ Stig Ekestorm, Sep -99
% Modified version of simhwchkv0.m
% Modifications will perform "parallel processing" and then "serial
summation" including:
% - correction at start-up ("initialize outputs from the taps, one tap after
another")
% - correction at the end ("shutting down the taps, one tap after another")
% Original file: simhwchk.m by SY YEO, 9 Jul 98
%
% Purpose: This program performs a architectural true simulation of the
% Digital Image Synthesizer hardware
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear

global dp_pts
global rg_pts
global doppler_inc

set(0,'defaultAxesFontSize',8);

noPlot = 0;
Ncontours = 20;

depthLUT = 32;
widthLUTFile = 2; % in units of number of hex digits
widthLUT = 8;  % n bits
ndopbits = 5;

%*****************************
% Read from data files
%*****************************
% Read from para.dat
fid = fopen('para.txt','r'); %opens para.txt to be read
tmp = fscanf(fid,'%d'); %reads in the values
nRangeCell = tmp(1); %1st value: 62, represents the number of range cells
nDopplerCell = tmp(2); %2nd value: 64, represents the number of radar pulses
targetExtent = tmp(3); %3rd value: 3, represents the radial length of the
target expressed in number of range cells
gain = tmp(4:4+targetExtent-1); %4th to 6th values: 1,2,4 - the gain value
for each tap
gainRev = fliplr(gain);
tmp1 = tmp(4+targetExtent:end); % 7th to last value: the phase increment
values for each tap
phi = reshape(tmp1,targetExtent,nDopplerCell); %3x64 matrix with zeros in the
1st column
fclose(fid);

% Read from rawint.dat
raw = zeros(nDopplerCell,nRangeCell); %create a 64x62 matrix, initialized to
zeros
fid = fopen('rawint.txt','r'); %open rawint.txt to be read
```

```
for j = 1:nDopplerCell
   for k = 1:nRangeCell-1
      raw(j,k) = fscanf(fid,'%d',1);
      comma = fscanf(fid,'%c',1);
   end
   raw(j,nRangeCell) = fscanf(fid,'%d',1);
end
fclose(fid);
[row,col] = size(raw);
raw = [raw,zeros(row,targetExtent-1)]; %raw: 64x64 matrix, last 2 columns
with zeros
%raw = [raw,zeros(row,targetExtent)]; %raw: 64x65 matrix, last 3 columns with
zeros

% Read from the LUT files.
load -ascii cosine.txt  % variable is cosine
load -ascii sine.txt    % variable is sine

% initialize some intermediate variables and vectors
Tgt_Extent=targetExtent;
DRFM_Phase=raw;
GainRev = gain';
Phase_inc=phi;
phiRev = zeros(Tgt_Extent,1);
depthLUT = 32;
phaseAdderOut = zeros(Tgt_Extent,1);
lutOut = zeros(Tgt_Extent,1);
tapOut = zeros(nRangeCell + (Tgt_Extent-1),Tgt_Extent);

% open files to write results to
f1 = fopen('checkv2.txt','w');   % "scan-path test"
f2 = fopen('Iout.txt','w');      % I-values, final output
f3 = fopen('Qout.txt','w');      % Q-values, final output
f4 = fopen('Iout_bin.txt','w');  % I-values in 2-complement binary, final
output
f5 = fopen('Qout_bin.txt','w');  % Q-values in 2-complement binary, final
output

% signal processing
for batchCnt = 1:nDopplerCell,

   fprintf(f1,'%s','*********************************************');
   fprintf(f1,'\r\n');
   fprintf(f1,'\r\n');
   fprintf(f1,'%s:%d\r\n','batchCnt (Radar Pulse)',batchCnt);
   fprintf(f1,'\r\n');
   disp(['Processing Pulse 'num2str(batchCnt)]);

   for intraPulseCnt = 1:(nRangeCell + (Tgt_Extent-1)), % clock cycle

      fprintf(f1,'%s:%d\r\n','intraPulseCnt',intraPulseCnt);
      fprintf(f1,'\r\n');
      fprintf(f1,'\t\t\t\t\t\t\t\t\t\t\t%s:%d\r\n','Clock
Pulse',intraPulseCnt);

      % --- This part simulates the intra pulse processing in hardware
```

```
%This part does "parallel processing" and then "serial summation"

% "parallel processing"

% initialize some intermediate variables and vectors
tap=zeros(1,Tgt_Extent);

% extraxt DRFM phase data
DRFM_data=DRFM_Phase(batchCnt,intraPulseCnt);
fprintf(f1,'%s\r\n','DRFM Phase Data processed:');
for idx=1:Tgt_Extent,
   tap(idx)=DRFM_data;
end
for idx=1:Tgt_Extent,
   fprintf(f1,'%s%d','   tap',idx-1);
   DRFM_bin=dec2two(DRFM_data,5,0);
   fprintf(f1,':%d\r\n',DRFM_data);
   fprintf(f1,' %d',DRFM_bin(1,2:6));
   fprintf(f1,'\r\n');
   fprintf(f1,'\r\n');
end

% phase addition (add phase increment (doppler offset) to DFRM phase
data)
phaseAdderOut=tap(1:Tgt_Extent)' + Phase_inc(:,batchCnt);
fprintf(f1,'\r\n');
fprintf(f1,'%s\r\n','Phase Increment:');
for idx=1:Tgt_Extent,
   fprintf(f1,'%s%d','   tap',idx-1);
   fprintf(f1,':%d\n',doppler_inc(idx));
   fprintf(f1,' %d',dec2two(doppler_inc(idx),3,0));
   fprintf(f1,'\r\n');
   fprintf(f1,'\r\n');
end
fprintf(f1,'\r\n');
fprintf(f1,'%s\r\n','Phase Rotation Register:');
for idx=1:Tgt_Extent,
   fprintf(f1,'%s%d','   tap',idx-1);
   fprintf(f1,':%d\n',mod(Phase_inc(idx,batchCnt),32));
   Pi=dec2two(mod(Phase_inc(idx,batchCnt),32),5,0);
   fprintf(f1,' %d',Pi(1,2:6));
   fprintf(f1,'\r\n');
   fprintf(f1,'\r\n');
end
fprintf(f1,'\r\n');
fprintf(f1,'\t\t\t\t\t\t\t\t\t\t\t%s:%d\r\n','Clock
Pulse',intraPulseCnt+1);
fprintf(f1,'%s\r\n','After Phase Adder:');
for idx=1:Tgt_Extent,
   fprintf(f1,'%s%d','   tap',idx-1);
   fprintf(f1,':%d\n',mod(phaseAdderOut(idx),32));
   pAO=dec2two(mod(phaseAdderOut(idx),32),5,0);
   fprintf(f1,' %d',pAO(1,2:6));
   fprintf(f1,'\r\n');
   fprintf(f1,'\r\n');
end
fprintf(f1,'\r\n');
```

```
      fprintf(f1,'\t\t\t\t\t\t\t\t\t\t%s:%d\r\n','Clock
Pulse',intraPulseCnt+2);

      % phase-amplitude look-up (to obtain complex time signal)
      tmp = mod(phaseAdderOut,depthLUT) + 1;
      lutOut = cosine(tmp) + sqrt(-1)*sine(tmp);
      fprintf(f1,'%s\r\n','After LUT (I-values):');
      for idx=1:Tgt_Extent,
         fprintf(f1,'%s%d','   tap',idx-1);
         fprintf(f1,':%5.7f\n',real(lutOut(idx)));
         fprintf(f1,' %d',dec2two(real(lutOut(idx)),0,7));
         fprintf(f1,'\r\n');
         fprintf(f1,'\r\n');
      end
      fprintf(f1,'%s\r\n','After LUT (Q-values):');
      for idx=1:Tgt_Extent,
         fprintf(f1,'%s%d','   tap',idx-1); .
         fprintf(f1,':%5.7f\n',imag(lutOut(idx)));
         fprintf(f1,' %d',dec2two(imag(lutOut(idx)),0,7));
         fprintf(f1,'\r\n');
         fprintf(f1,'\r\n');
      end

      % correction at the end ("shutting down the taps one tap after
another")
      if intraPulseCnt>nRangeCell,
         for idx2=1:(intraPulseCnt-nRangeCell),
            lutOut((idx2),:)=0;
         end
      end

      fprintf(f1,'\r\n');
      fprintf(f1,'\t\t\t\t\t\t\t\t\t\t%s:%d\r\n','Clock
Pulse',intraPulseCnt+3);

      % gain modulation, and storing values in an intermediate matrix
      fprintf(f1,'%s\r\n','Gain Value:');
      for idx=1:Tgt_Extent,
         if GainRev(1,idx)==1,
            GainRev2=0;
         elseif GainRev(1,idx)==2,
            GainRev2=1;
         elseif GainRev(1,idx)==4,
            GainRev2=2;
         elseif GainRev(1,idx)==8,
            GainRev2=3;
         end
         fprintf(f1,'%s%d','   tap',idx-1);
         fprintf(f1,':%5.7f\n',GainRev(1,idx));
         GRbin=dec2two(GainRev2,2,0);
         fprintf(f1,' %d',GRbin(1,2:3));
         fprintf(f1,'\r\n');
         fprintf(f1,'\r\n');
      end
      if intraPulseCnt<=nRangeCell,
         GainOut = GainRev.*lutOut';
         for idx3=0:Tgt_Extent-1,
```

```
            tapOut(intraPulseCnt+idx3,idx3+1)=GainOut(idx3+1);
         end
      end
      fprintf(f1,'%s\r\n','After Gain Block (I-values):');
      for idx=1:Tgt_Extent,
         fprintf(f1,'%s%d','   tap',idx-1);
         fprintf(f1,':%5.7f\n',real(GainOut(idx)));
         fprintf(f1,' %d',dec2two(real(GainOut(idx)),3,7));
         fprintf(f1,'\r\n');
         fprintf(f1,'\r\n');
      end
      fprintf(f1,'%s\r\n','After Gain Block (Q-values):');
      for idx=1:Tgt_Extent,
         fprintf(f1,'%s%d','   tap',idx-1);
         fprintf(f1,':%5.7f\n',imag(GainOut(idx)));
         fprintf(f1,' %d',dec2two(imag(GainOut(idx)),3,7));
         fprintf(f1,'\r\n');
         fprintf(f1,'\r\n');
      end

      fprintf(f1,'\r\n');
      fprintf(f1,'\t\t\t\t\t\t\t\t\t\t\t%s:%d\r\n','Clock
Pulse',intraPulseCnt+4);

      % final accumulation - "serial summation"
      % - 1st: extract partial sums (I and Q)
      % - 2nd: extract final sums (I and Q)
      tapNew=tapOut;
      add=0;
      tt=Tgt_Extent;
      fprintf(f1,'%s\r\n','Final values in each tap (I- and Q-values):');
      fprintf(f1,'%s%d','   Final I-value in tap',tt-1);
      fprintf(f1,':%5.7f\n',real(tapNew(intraPulseCnt,tt)));
      fprintf(f1,' %d',dec2two(real(tapNew(intraPulseCnt,tt)),8,7));
      fprintf(f1,'\r\n');
      fprintf(f1,'\r\n');
      fprintf(f1,'%s%d','   Final Q-value in tap',tt-1);
      fprintf(f1,':%5.7f\n',imag(tapNew(intraPulseCnt,tt)));
      fprintf(f1,' %d',dec2two(imag(tapNew(intraPulseCnt,tt)),8,7));
      fprintf(f1,'\r\n');
      fprintf(f1,'\r\n');

      while tt>=2,
         add=add+1;
         tapNew(intraPulseCnt,tt-
1)=tapNew(intraPulseCnt,tt)+tapNew(intraPulseCnt,tt-1);
         fprintf(f1,'%s%d','   Final I-value in tap',tt-2);
         fprintf(f1,':%5.7f\n',real(tapNew(intraPulseCnt,tt-1)));
         fprintf(f1,' %d',dec2two(real(tapNew(intraPulseCnt,tt-1)),8,7));
         fprintf(f1,'\r\n');
         fprintf(f1,'\r\n');
         fprintf(f1,'%s%d','   Final Q-value in tap',tt-2);
         fprintf(f1,':%5.7f\n',imag(tapNew(intraPulseCnt,tt-1)));
         fprintf(f1,' %d',dec2two(imag(tapNew(intraPulseCnt,tt-1)),8,7));
         fprintf(f1,'\r\n');
         fprintf(f1,'\r\n');
         partial_tapsum(intraPulseCnt,1)=tapNew(intraPulseCnt,tt-1);
```

```
            tt=tt-1;
        end

        if tt==1,
            partial_tapsum(intraPulseCnt,1)=tapOut(intraPulseCnt,1);
        end

        fprintf(f1,'\r\n');
        Iout=real(partial_tapsum(intraPulseCnt,1));
        Qout=imag(partial_tapsum(intraPulseCnt,1));
        fprintf(f1,'%s\r\n','Final Output values (I- and Q-values):');
        fprintf(f1,'%s%d','   Iout - Final I-value for
intraPulseCnt',intraPulseCnt);
        fprintf(f1,':%5.7f\n',Iout);
        fprintf(f1,' %d',dec2two(Iout,8,7));
        fprintf(f1,'\r\n');
        fprintf(f1,'\r\n');
        fprintf(f1,'%s%d','   Qout - Final Q-value for
intraPulseCnt',intraPulseCnt);
        fprintf(f1,':%5.7f\n',Qout);
        fprintf(f1,' %d',dec2two(Qout,8,7));
        fprintf(f1,'\r\n');
        fprintf(f1,'\r\n');
        fprintf(f1,'%s','-----------------------------------------------');
        fprintf(f1,'\r\n');

        % write final results (I and Q) to separate files
        fprintf(f2,'%5.7f\n',Iout);
        fprintf(f3,'%5.7f\n',Qout);
        fprintf(f4,'%d',dec2two(Iout,8,7));
        fprintf(f4,'\r\n');
        fprintf(f5,'%d',dec2two(Qout,8,7));
        fprintf(f5,'\r\n');

    end %intraPulseCnt

    finalAdderOut(batchCnt,:)=partial_tapsum';

end %batchCnt

% close files
fclose(f1);
fclose(f2);
fclose(f3);
fclose(f4);
fclose(f5);


%*****************************
% Pulse Compression
%*****************************
%--- Compress the doppler shifted signals
load pc_ref
priRgMapShift = zeros(nDopplerCell,rg_pts);
tic
pcRefMapShift = fft(finalAdderOut.',2*rg_pts-1).';
for idx = 1:nDopplerCell
```

```
    tmp = cref.*pcRefMapShift(idx,:);
    tmp1 = fftshift(ifft(tmp));
    priRgMapShift(idx,1:end-targetExtent+1) =  tmp1(rg_pts+targetExtent-
1:end);
end
dpRgMapShiftMOD = abs(fft(priRgMapShift));
%dpRgMapShift = abs(fft(priRgMapShift));
toc

save plotMOD dpRgMapShiftMOD

save fAddOut finalAdderOut

%*****************************
% Display
%*****************************
if (noPlot == 0)
    figure(2);
    load plot.mat
    subplot(2,1,1);
    h = contour(dpyq,Ncontours); grid; axis([1 62 0 dp_pts])
    %h = contour(dpyq_shift_drfm,Ncontours); grid; axis([0 20 0 32])
    title('a. Amplitude and Doppler Modulated Rd-Dp Map (unmodulated / MATLAB)
');
    xlabel('Down Range Cells');   ylabel('Cross Range Cells');
    axis([1 62 0 dp_pts])
    subplot(2,1,2);
    h = contour(dpRgMapShiftMOD,Ncontours); grid; axis([1 62 0 dp_pts])
    %h = contour(dpRgMapShift,Ncontours); grid; axis([1 20 0 32])
    title('b. Amplitude and Doppler Modulated Rd-Dp Map (Bit-True, modulated /
MATLAB) ');
    xlabel('Down Range Cells');   ylabel('Cross Range Cells');
    axis([1 62 0 dp_pts])
end
```

plothwv2.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% plothwv2.m
% MAJ Stig Ekestorm, Sep -99
% Modified version of plothwv1.m by Stig Ekestorm, Aug -99
% Original file: plothwv0.m by SY YEO, 11 Aug 1998
%
% This version processes the output from the LUT
% Works in concert with mathostv2.m and simhwchkv2.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear

global hda
global dp_pts
global rg_pts

set(0,'defaultAxesFontSize',7)

noplot = 0;

%to load data from hardware output files
if hda==1,
   load -ascii imagei.txt
   load -ascii imageq.txt
end

fid = fopen('para.txt','r');
tmp = fscanf(fid,'%d');
nRangeCell = tmp(1);
nDopplerCell = tmp(2);
targetExtent = tmp(3);
fclose(fid);

%for getting the data form hardware in the right format
if hda==1,
   image = reshape(image,nRangeCell+(targetExtent-1),nDopplerCell);
   image = imagei - j*imageq;
   image = reshape(image,nRangeCell+(targetExtent-1),nDopplerCell); %for ASIC
simulation
   %image = reshape(image,nRangeCell+targetExtent,nDopplerCell); %for FPGA 3-
tap simulation
   image = image';
end

load fAddOut

if (noplot == 0)
   %*****************************
   % Pulse Compression
   %*****************************
   %--- Compress the doppler shifted signals
   figure(3);
   orient tall
```

```
   load plot.mat
   load plotMOD.mat

   Ncontours = 9;
   subplot(2,1,1);
   %h = contour(dpyq_shift_drfm,Ncontours); grid, axis([1 62 0 64])
   h = contour(dpRgMapShiftMOD,Ncontours); grid; axis([1 62 0 dp_pts])
   title('a. Amplitude and Doppler Modulated Rd-Dp Map (Bit and Architecture-
True / MATLAB)');
   xlabel('Down Range Cells');   ylabel('Cross Range Cells');

   %to post-process data from hardware
   if hda==1,
      load pc_ref
      priRgMapShift = zeros(nDopplerCell.',rg_pts);
      tic
      pcRefMapShift = fft(image.',2*rg_pts-1).';
      for idx = 1:nDopplerCell
         tmp = cref.*pcRefMapShift(idx,:);
         tmp1 = fftshift(ifft(tmp));
         priRgMapShift(idx,1:end-targetExtent+1) =  tmp1(rg_pts+targetExtent-
1:end);
      end
      dpRgMapShift = abs(fft(priRgMapShift));
      toc
   end

   subplot(2,1,2);
   if hda==1,
      h = contour(dpRgMapShift,Ncontours);
      grid, axis([1 62 0 dp_pts])
   end
   title('b. Amplitude and Doppler Modulated Rd-Dp Map (from HARDWARE
output)');
   xlabel('Down Range Cells');   ylabel('Cross Range Cells');
end

figure(3)
print -dtiff hwres

figure(4)
subplot(3,1,1);
h = mesh(dpRgMapShiftMOD); grid;
%h = mesh(dpyq_shift_drfm); grid;
title('a. Amplitude/Doppler Modulated Rd-Dp Map (Bit-True, modulated /
MATLAB)');
xlabel('Down Range Cells');   ylabel('Cross Range Cells'); grid
subplot(3,1,2);
%to plot hardware output
if hda==1,
   h = mesh(dpRgMapShift); grid;
end
title('b. Amplitude/Doppler Modulated Rd-Dp Map (HARDWARE output)');
xlabel('Down Range Cells');   ylabel('Cross Range Cells'); grid
subplot(3,1,3);
%to plot difference between Matlab simulation and hardware output
if hda==1,
```

```
   %h = mesh(dpyq_shift_drfm/max(max(dpyq_shift_drfm))-
dpRgMapShift/max(max(dpRgMapShift))); grid;
   h = mesh(dpRgMapShiftMOD-dpRgMapShift); grid; %plot the real difference,
Stig Aug-99
   %h = mesh(dpyq_shift_drfm-dpRgMapShift); grid; %plot the real difference,
Stig Aug-99
end
title('c. Difference');
xlabel('Down Range Cells');   ylabel('Cross Range Cells'); grid
print -dtiff diffplot


%for comparison 5 Oct -99, Stig Ekestorm
%to plot MATLAB simulation output separately
figure(5)
h = mesh(dpRgMapShiftMOD); grid;
title('a. Amplitude/Doppler Modulated Rd-Dp Map (Bit-True, modulated /
MATLAB)');
xlabel('Down Range Cells');   ylabel('Cross Range Cells'); grid

figure(6);
%to plot hardware output separately
if hda==1,
   h = mesh(dpRgMapShift); grid;
end
title('b. Amplitude/Doppler Modulated Rd-Dp Map (HARDWARE output)');
xlabel('Down Range Cells');   ylabel('Cross Range Cells'); grid
```

dec2two.m

```
%*********************************************************************
%dec2two.m
%
%This MATLAB function converts a number in decimal representation
%(positive or negative) to a vector in binary 2-complement representation.
%With a slight modification the output can be presented in as a string with a
%"." character separating integer and fractional parts. The user has to
specify
%the number to be converted and the format for the %binary presentation
(number
%of bits used for the integer part and the %fractional part). A sign bit will
%automatically be included in the output vector (string).
%
%Function call:
%   dec2two(dec,integerbits,fractionbits)
%
%User inputs:
%   dec - the number in decimal representation to be converted
%   integerbits - # of bits to represent the integer part
%   fractionbits - # of bits to represent the fractional part
%
%Example (1):
%   type in the MATLAB Command Window: dec2two(2.75,8,4)
%   returned answer: 0  0  0  0  0  0  0  1  0  1  1  0  0
%   (returned answer: 0  0  0  0  0  0  0  1  0 . 1  1  0  0)
%
%Example (2):
%   type in the MATLAB Command Window: dec2two(-2.75,8,4)
%   returned answer: 1  1  1  1  1  1  1  0  1  0  1  0  0
%   (returned answer: 1  1  1  1  1  1  1  0  1 . 0  1  0  0)
%
%Created by:
%   MAJ Stig Ekestorm, 3 Oct -99
%   Naval Postgraduate School
%
%*********************************************************************

function [out] = dec2two(dec,integerbits,fractionbits);

%binary format to be displayed
signbit=1;                                  %# of bits to represent the
sign
%integerbits                                %# of bits to represent the
integer part
%fractionbits                               %# of bits to represent the
fractional part

%initialize output vector
aa=signbit+integerbits+fractionbits;        %length of output vector
bb=zeros(1,aa);                             %initialize output vector to
zero

%check if the number is negative
if dec<0,                                   %if negative number
```

```
   dec=dec*(-1);                                    %turn number into positive
   bb(1,1)=1;                                       %set sign bit to "1"
end                                                 %end if statement

%integer part
mm=floor(dec);                                      %extract the integer part
for idx1=2:integerbits+1,                           %integer bits (sign bit not
included)
   cc=2^(integerbits+1-idx1);                       %binary bit representation
   bb(1,idx1)=floor(mm/cc);                         %set each bit after integer
division
   mm=rem(mm,cc);                                   %extract reminder after
division
end                                                 %end for loop

%fractional part
ff=dec-floor(dec);                                  %extract fractional part
for idx2=signbit+integerbits+1:aa,                  %fraction bits
   dd=1/(2^(idx2-(signbit+integerbits)));           %binary bit representation
   bb(1,idx2)=floor(ff/dd);                         %set each bit after integer
division
   ff=rem(ff,dd);                                   %extract reminder after
division
end                                                 %end for loop

%adjust negative value to 2-complement representation
if bb(1,1)==1,                                      %if negative value
   for idx3=1:aa,                                   %index for switching bit
values
      if bb(1,idx3)==2,
         bb(1,idx3)=0;
      end
      if bb(1,idx3)==0,                             %switch all "0"
         bb(1,idx3)=1;                              %to "1"
      else
         bb(1,idx3)=0;                              %and vice versa
      end                                           %end if/else statement
   end                                              %end for loop
   bb(1,aa)=bb(1,aa)+1;                             %add "1" to the LSB
   idx4=aa;                                         %index for binary addition
   while bb(1,idx4)==2,                             %if carry (bit-to-bit)
      bb(1,idx4-1)=bb(1,idx4-1)+1;                  %add "1" to next higher bit
      bb(1,idx4)=0;                                 %set current bit to "0"
      idx4=idx4-1;                                  %decrement index (higher bit)
   end                                              %end while loop
   bb(1,1)=1;
end                                                 %end if/else statement

%return output in string format (integer and fractional parts separated by
".")
%out=([num2str(bb(1,1:signbit+integerbits)),' . '...
%      ,num2str(bb(1,signbit+integerbits+1:aa))]);
%return output in vector format (integer and fractional parts without
separation)
out=bb;
%end of file
```

two2dec.m

```
%*********************************************************************
%two2dec.m
%
%This MATLAB function convert vectors in binary 2-complement representation
%to numbers in decimal representation (positive and negative).
%The user has to specify a vector or a set of vectors in a matrix to be
%converted, and the format for the binary presentation (number of bits used
%for the fractional part). The first bit is assumed to be a sign bit for the
%binary vector.
%
%Function call:
%   two2dec(two,fractionbits)
%
%User inputs:
%   two          - the vector/matrix of vectors in binary 2-complement
%                  representation to be converted to decimal number
%   fractionbits - # of bits that represent the fractional part
%
%Output:
%   number/set of numbers in decimal representation
%
%Created by:
%   MAJ Stig Ekestorm, 6 Nov -99
%   Naval Postgraduate School
%*********************************************************************

%function call
function [out] = two2dec(two,fractionbits);

%determine the size of the matrix of vectors to be converted
[row col]=size(two);

%initialize vectors/variables used
integer=two(:,1:col-fractionbits);  %integer part
fraction=two(:,col-fractionbits+1:col);   %fractional part
[rowi coli]=size(integer); %size of integer part
[rowf colf]=size(fraction);   %size of fractional part
out=zeros(row,col/col); %output vector

%convert bit pattern representing binary 2-complement number to decimal
number
for idx1=1:row,   %vector-by-vector

   %if positive number - convert
   if integer(idx1,1)==0,  %if sign bit is "0" (positive)
      testi=fliplr(integer(idx1,1:coli)); %flip integer part of vector
      testf=fraction(idx1,1:colf); %fractional part of vector
      for idx2=1:coli-1,   %integer part, bit-by-bit
         if testi(1,idx2)==1, %check for ones
            out(idx1,1)=out(idx1,1)+2^(idx2-1); %add decimal value
         end   %end if
      end   %end for
      for idx2=1:colf, %fractional part, bit-by-bit
         if testf(1,idx2)==1, %check for ones
```

```
            out(idx1,1)=out(idx1,1)+2^(-(idx2));   %add decimal value
         end   %end if
      end   %end for

   %if negative number - adjust and convert
   else  %if sign bit is "1" (negative)
      for idx4=2:coli,  %index for switching integer bit values
         if integer(idx1,idx4)==0,  %switch all "0"
            integer(idx1,idx4)=1;   %to "1"
         else
            integer(idx1,idx4)=0;   %and vice versa
         end   %end if/else statement
      end   %end for loop
      if coli>1,  %if there is a integer part
         integer(idx1,coli)=integer(idx1,coli)+1;  %add "1" to the LSB
         idx5=coli;  %index for binary addition
         while integer(idx1,idx5)==2,  %if carry (bit-to-bit)
            integer(idx1,idx5-1)=integer(idx1,idx5-1)+1; %add "1" to next
higher bit
            integer(idx1,idx5)=0;   %set current bit to "0"
            idx5=idx5-1;   %decrement index (higher bit)
            if idx5==1, %if this is the last bit
               integer(idx1,idx5)=1;   %reset sign bit to "1"
            end   %end if
         end   %end while
      end   %end if
      testi=fliplr(integer(idx1,1:coli)); %flip integer part of vector
      testf=fraction(idx1,1:colf);  %fractional part of vector
      for idx2=1:coli-1,   %integer part, bit-by-bit
         if testi(1,idx2)==1, %check for ones
            out(idx1,1)=out(idx1,1)+2^(idx2-1); %add decimal value
         end   %end if
      end   %end for
      for idx2=1:colf, %fractional part, bit-by-bit
         if testf(1,idx2)==1, %check for ones
            out(idx1,1)=out(idx1,1)-2^(-(idx2));   %add decimal value
         end   %end if
      end   %end for
      out(idx1,1)=(1+out(idx1,1))*(-1);   %assign negative value
      %if only sign bit and all interger bits are "0", and there are fraction
bits
      if isempty(find(integer(idx1,2:coli)))==1 & find(fraction)>0,
         out(idx1,1)=(out(idx1,1)+2^(coli-1))*(-1)-1; %adjust value
      end
      if find(fraction)>0, %if there are fraction bits
         out(idx1,1)=out(idx1,1)+1; %do nothing
      else  %if no fraction bits
         out(idx1,1)=out(idx1,1)+1; %adjust value
      end   %end if
      %if only the sign bit is "1" and all other bits "0"
      if isempty(find(fraction))==1 & find(integer(idx1,:))<2,
         out(idx1,1)=(out(idx1,1)+2^(coli-1))*(-1);   %adjust value
      end   %end if
   end   %end if
end   %end for
%end of file
```

MATLAB Code – Cosine and Sine Look-Up Tables (txt-files)

cosine.txt

```
 9.9218750e-001
 9.6875000e-001
 9.0625000e-001
 8.1250000e-001
 6.7968750e-001
 5.2343750e-001
 3.4375000e-001
 1.4843750e-001
-4.6875000e-002
-2.4218750e-001
-4.2968750e-001
-6.0156250e-001
-7.5000000e-001
-8.6718750e-001
-9.4531250e-001
-9.8437500e-001
-9.8437500e-001
-9.4531250e-001
-8.6718750e-001
-7.5000000e-001
-6.0156250e-001
-4.2968750e-001
-2.4218750e-001
-4.6875000e-002
 1.4843750e-001
 3.4375000e-001
 5.2343750e-001
 6.7968750e-001
 8.1250000e-001
 9.0625000e-001
 9.6875000e-001
 9.9218750e-001
```

sine.txt

```
 0.0000000e+000
 1.9531250e-001
 3.9062500e-001
 5.6250000e-001
 7.1875000e-001
 8.3593750e-001
 9.2968750e-001
 9.7656250e-001
 9.8437500e-001
 9.5312500e-001
 8.9062500e-001
 7.8125000e-001
 6.4062500e-001
 4.7656250e-001
 2.9687500e-001
 9.3750000e-002
-9.3750000e-002
-2.9687500e-001
-4.7656250e-001
-6.4062500e-001
-7.8125000e-001
-8.9062500e-001
-9.5312500e-001
-9.8437500e-001
-9.7656250e-001
-9.2968750e-001
-8.3593750e-001
-7.1875000e-001
-5.6250000e-001
-3.9062500e-001
-1.9531250e-001
 0.0000000e+000
```

Appendix B - Visual Basic Codes

fileio (file.bas)

```
Public nRangeCell As Integer
Public nDopplerCell As Integer
Public targetExtent As Integer
Public Gain() As Integer
Public Phi() As Integer
Public DRFM() As Integer
Public sine() As Double
Public cosine() As Double

Public Sub readPara()
    ' This sub-routine reads the processing parameters generated by Matlab
(reading from paraVB.txt)
    Dim idx As Integer
    Dim idx1 As Integer

    Open "c:/temasek/denise/thesis/final_design/vbfiles/para.txt" For Input
As #1

    ' read in number of range cells (number of samples of the chirp signal)
    Input #1, nRangeCell
    flexform.ParaText(0) = Val(nRangeCell)

    ' read in number of doppler cells (ndop) first
    Input #1, nDopplerCell
    flexform.ParaText(1) = Val(nDopplerCell)

    ' next read in the target extent
    Input #1, targetExtent
    flexform.ParaText(2) = Val(targetExtent)

    ' read in gain values (number of gain values = targetExtent)
    ' ReDim Gain(targetExtent - 1) As Integer
    ReDim Gain(3) As Integer
    Gain(0) = 0
    Gain(1) = 0
    Gain(2) = 0
    For idx = 0 To targetExtent - 1
        Input #1, Gain(idx)
        ' adjustment to correct multiplication factors for the amplitude
(gain) value
        Select Case Gain(idx)
            Case Is = 1
            Gain(idx) = 0    ' no shift, multiplication by 1, hardware bit
"00"
            Case Is = 2
            Gain(idx) = 1    ' shift by 1, multiplication by 2, hardware bit
"01"
            Case Is = 4
            Gain(idx) = 2    ' shift by 2, multiplication by 4, hardware bit
"10"
            Case Is = 8
```

```
            Gain(idx) = 3    ' shift by 3, multiplication by 8, hardware bit
"11"
        End Select
    Next idx

    ' read in the next nDoppler values
    ReDim Phi(nDopplerCell - 1, targetExtent - 1) As Integer
    For idx = 0 To nDopplerCell - 1
        For idx1 = 0 To targetExtent - 1
            Input #1, Phi(idx, idx1)
        Next idx1
    Next idx
    Close #1
End Sub
Public Sub readRaw()
    ' This sub-routine reads the raw matlab-simulated ISAR data

    Dim idx1 As Integer
    Dim idx2 As Integer

    Open "c:/temasek/denise/thesis/final_design/vbfiles/rawint.txt" For Input
As #1

    ' Create the array dynamically
    ReDim DRFM(nDopplerCell - 1, nRangeCell + targetExtent - 1) As Integer

    ' read in the first samples of the first pulse
    ' these values are phase values from the DFRM
    For idx1 = 0 To nDopplerCell - 1
        For idx2 = 0 To nRangeCell - 1
            Input #1, DRFM(idx1, idx2)
        Next idx2
    Next idx1
    Close #1
 End Sub
Public Sub readCosine()
    ' This sub-routine reads the raw matlab-simulated ISAR data

    Dim idx1 As Integer
    Dim tmp As String

    Open "c:/temasek/denise/thesis/final_design/vbfiles/cosine.txt" For Input
As #1

    ' Create the array dynamically
    ReDim cosine(32) As Double

    ' read in the first samples of the first pulse
    ' these values are phase values from the DFRM
    For idx1 = 0 To 31
        Line Input #1, tmp
        cosine(idx1) = Val(tmp)
    Next idx1
    Close #1
 End Sub
Public Sub readSine()
    ' This sub-routine reads the raw matlab-simulated ISAR data
```

```
    Dim idx1 As Integer

    Open "c:/temasek/denise/thesis/final_design/vbfiles/sine.txt" For Input
As #1

    ' Create the array dynamically
    ReDim sine(32) As Double
    Dim tmp As String

    ' read in the first samples of the first pulse
    ' these values are phase values from the DFRM
    For idx1 = 0 To 31
        Line Input #1, tmp
        sine(idx1) = Val(tmp)
    Next idx1
    Close #1
 End Sub
```

FlexFunc (Flexfunc.bas)

```
Option Explicit
'U4 Flex Programming Logic Control & Status Ports
'   Control
'   Port 380/382 - Write          A/B          A/B
'       D0  =   nConfig     U4-H10/B8  Flex U27/U28 - AU1
'       D1  =   nCS         U4-J11/F9  Flex U27/U28 - A35
'                CS                    Flex U27/U28 - C33 (HI)
'       D2  =   nClr        U4-D11/F3  Flex U27/U28 - C17
'       D3  =   OutEn       U4-K3/K10  Flex U27/U28 - C19
'3
'
'   Port 380/382 - Read
'       D4 =    Init_Done   U4-L3/L2   Flex U27/U28 - R35
'       D5 =    Conf_Done   U4-L4/K4   Flex U27/U28 - C37
'       D6 =    nStatus     U4-L10/L9  Flex U27/U28 - AU37
'       D7 =    RDYnBSY     U4-J2/K11  Flex U27/U28 - N35
'
'   Port 381/383 - Write
'       D0-D7   Configuration Data for Flex chip A/B
'               nWS         U4-H11/E11 Flex U27/U28 - E31
'
'   Port 381/383 - Read
'       D7 -    RDYnBSY for Flex chip A/B
'               nRS         U4-G9/F10  Flex U27/U28 - A33
'
'
'   Port 384 -          FLEX User Control/Status Register
'       D0  = Enable DATA_DIR for READ Buffers (D0 on U4-A9, DATA_DIR on U4-
B11)
'       D1  = An/B  Select Flex A = 0, Select Flex B = 1
'       D2  = Int_CLRn
'       D3  = Unused
'       D4  = CFA1  Spare chip interconnect to FLEX A
'       D5  = CFB1  Spare chip interconnect ot FLEX B
'       D6  = TST_A Flex A circuit Test Point
'       D7  = TST_B FLEX B circuit TEst point
'
'
'   Port 386 -          FLEX A (left from component side) User Base Addr.
Register
'   Port 387 -          FLEX B (right from component side) User Base Addr.
Register

Declare Function write_port Lib "in_out" (paddr%, pdata%, byteword%) As
Integer
Declare Function read_port Lib "in_out" (paddr%, pdata%, byteword%) As
Integer
Declare Function InitClicks Lib "in_out" () As Integer
Declare Function Clicks Lib "in_out" () As Integer
Declare Function disable_int Lib "in_out" () As Integer
Declare Function enable_int Lib "in_out" () As Integer

Declare Function FlexConfig Lib "altera" (ByVal file As String, ByRef plength
As Long) As Integer
```

```
Declare Function FlexSend Lib "altera" (ByVal Cntrl_port%, ByVal
Status_port%, ByVal Data_port%) As Integer
Declare Function FlexSend10k50 Lib "altera" (ByVal Cntrl_port%, ByVal
Status_port%, ByVal Data_port%) As Long

'Memory Calls
Declare Function MemoryInit Lib "memory" (ByVal Start As Long, ByVal Length
As Long) As Integer
Declare Function MemoryRead Lib "memory" (ByVal Location As Long, Value As
Any) As Integer
Declare Function MemoryWrite Lib "memory" (ByVal Location As Long, ByVal
Value As Long) As Integer
Declare Function MemoryReadBuffer Lib "memory" (ByVal Location As Long, ByVal
Count As Long, ByRef Value As Integer) As Integer
Declare Function MemoryWriteBuffer Lib "memory" (ByVal Location As Long,
ByVal Count As Long, ByRef Value As Integer) As Integer

'LoadTgt DLL
Declare Function LoadTgtBuf Lib "loadtgt" (ByVal TgtNum%, ByVal param%, ByVal
Value&) As Integer
Declare Function TargetWrite Lib "loadtgt" (ByVal TgtNum%) As Integer
Declare Function WriteTargets Lib "loadtgt" (ByVal NumTgts%) As Integer
Declare Function InitializePorts Lib "loadtgt" () As Integer

'*** RES added variables ***
Public NumBds As Integer, offset As Integer
Public flex_user_ba_ctrl As Integer     ' set this port to determine user
design ba
Public user_ba(8) As Integer
Public BoardNum As Integer
Dim maxcount As Integer
Public present(8) As Integer, fstatus(8) As Integer
Public configdone(8) As Integer, Status(8) As Integer
Public num_bds_fnd As Integer
'***************************

' RMS 17 Sep 96
' Do I need to declare Function Delay(Dwell As Double) ?

Global filename As String
Global MSG As Integer
Global BoardType(8) As String      'Either "10K" or "8K"

' 5032 or 5192 Chip Addresses
Global FlexCtrlPortBA(8) As Integer
Global FlexCtrlPort As Integer
Global FlexStatusPort As Integer
Global FlexDataPort As Integer
Global FlexUserCtrlPort As Integer  'Used to Toggle between chips
Global FlexUserBasePort As Integer  'Used to Set base address in Flex
Global NumFlexes As Integer
Global FlexIndex As Integer
Global NumFlexFiles As Integer
Global FlexFileIndex As Integer
'RMS 17 Sep 96
'These are for 8K only, and maybe not there if we change the 5192
Global FlexONPort As Integer
```

```
Global FlexOFFPort As Integer

'Flex 10K50 Chip Addresses
Global FlexUserBA(8) As Integer

'Flex File Names and Documentation
Global flexfilename(10) As String
Global FlexMaxFiles As Integer
Global FlexFileDoc(10) As String

Global DP_MEM_Right_Addr_Lo As Integer
Global DP_MEM_Right_Addr_HI As Integer
Global DP_MEM_Right_Data As Integer
Global DP_MEM_Right_Ctrl As Integer

Global DP_MEM_Left_Addr_Lo As Integer
Global DP_MEM_Left_Addr_HI As Integer
Global DP_MEM_Left_Data As Integer
Global DP_MEM_Left_Ctrl As Integer

Global DP_MEM_Hand_Shake_Sim As Integer

Global DP_MEM_Left_addr_Mux As Integer
Global HP_Ctrl_Port As Integer     'Control Port to select HP connector A
data
Global HP_Ctrl_data As Integer     'Control data to Select HP connector B
data
                                    ' Data - HPA     HPB
                                    '   0   Toggle  Toggle
                                    '   1   Mem     Toggle
                                    '   2   Toggle  Mem
                                    '   3   Mem     Mem
Global AttenPortLO As Integer
Global AttenPortHI As Integer

Global Const nConfigLo = &H2
Global Const nConfigHI = &H3
Global Const nConfigHI_nCSLO = &H1
Global Const nConfigHI_nCSHI = &H3
Global Const nStatLO_RDYnBSYHI = &HC
Global Const ConfDone = &H18            'Conf_Done & nSatus HI


Global altera(100000) As Integer
Global MaxAlteraPnts As Long

Global Const RngDef = &H100
Global Const PWDef = &H200

'***** internal addresses for the ISAR program *****
'Global Const phiAddr = &H10          ' for doppler offset
'Global Const gainAddr = &H20         ' for gain
'Global Const tapAddr = &H30          ' for tap delay line
'Global Const modPulseAddr = &H40     ' for modulated pulse readback
'Global Const feedback = &H60          ' for reading back values

Function Delay(Dwell As Double)
```

```
' This routine creates a time delay that lasts untill Dwell seconds
' elapse from the time of call.
'
' It uses VBasic's Timer function, which returns the number of seconds
' since midnight on the system clock, rolling from 86400 to 0 at midnight.
'
' This routine allows delays to begin before midnight and end after,
' or that span several days.

Dim SecPerDay As Double
Dim Start As Double, Done As Double, T As Double, LastT As Double

SecPerDay = 86400# ' = 60.0 * 60.0 * 24.0 seconds in a day

Start = Timer
Done = Start + Dwell

While (Done > SecPerDay) ' Midnight will come before the delay elapses.
    LastT = Start
    T = Timer
    While (T > LastT) ' Timer has not rolled over.
        LastT = T
        T = Timer
        DoEvents
    Wend
    ' It's midnight, so deduct the previous day's waiting
    ' and start a new day.
    Done = Done - (SecPerDay - Start)
    Start = 0
Wend

' The delay will elapse before midnight comes.
While (Done > Timer)
    DoEvents
Wend

End Function
Public Function LoadFlex(filename As String, Index As Integer)

Dim MSG As Long
Dim dum1 As Integer
Static Status As Integer
Dim FileDate As String

'setup flex addresses
    FlxBaseAddr (BoardNum)

'Reset Flex Chip
MSG = write_port(FlexCtrlPort, nConfigLo, 1)  'Set nConfig (bit 0) LO

'Read & Send Data'
DoEvents

    MSG = FlexConfig(filename, MaxAlteraPnts)

If (MSG = 1) Then
```

```
        MSG = FlexSend10k50(FlexCtrlPort, FlexStatusPort, FlexDataPort)
        Status = Get10KStatus()

    FileDate = FileDateTime(filename)
    If Status = True Then
       DoEvents
       MSG = write_port(FlexCtrlPort, &HF, 1)     'Set nCONFIG, nCS, nCLR,
IO_ENB = 1
       LoadFlex = True
    Else
       LoadFlex = False
     End If

    Delay (0.2)              'Delay .2 Second for Visual Effect

Else
    LoadFlex = False
End If

End Function
Public Sub InitFlexType()

Dim i As Integer
Dim ret As Integer
Dim memory_length As Long
Dim status_init As Integer

BoardType(BoardNum) = "10K"

  Call InitBoardBase(BoardType(BoardNum))     'Get Base Address

If BoardType(BoardNum) = "10K" Then InitFlx10KAddr (BoardNum)        'Set
First Board as Default

End Sub

Function Get10KStatus()
Static MSG As Integer

    MSG = write_port(FlexUserCtrlPort, &H1, 1)       'Turn on Read Buffer
Capability

    MSG = read_port(FlexStatusPort, fstatus(BoardNum), 1)         'Get Flex
Status

    If fstatus(BoardNum) = &HFF Then
      'the board is off or not plugged in
      Get10KStatus = False
      present(BoardNum) = False
      Status(BoardNum) = False
      configdone(BoardNum) = False
      Exit Function
    End If

    If (&H40 And fstatus(BoardNum)) <> &H40 Then
      'nSTATUS bit = 0 -- an error occurred
```

```
      Get10KStatus = False
      Status(BoardNum) = False
      configdone(BoardNum) = False
      Exit Function
    Else
   End If

   If (&H60 And fstatus(BoardNum)) = &H60 Then
     'nSTATUS bit = 1 and CONF_DONE = 1 -- the FLEX programmed OK
''     ConfigDone.Value = 1
     Else
''     ConfigDone.Value = 0
     configdone(BoardNum) = False
     Get10KStatus = False
     Exit Function
   End If

   present(BoardNum) = True
   Status(BoardNum) = True
   configdone(BoardNum) = True
   Get10KStatus = True
End Function
Sub get_flex_ini()
Dim a As String, b As Integer

   On Error GoTo ini_err_handler

   Open "flex.ini" For Input As #1
       Input #1, NumBds, a
     For b = 1 To NumBds
       Input #1, BoardNum
       Input #1, FlexCtrlPortBA(BoardNum), crystal_clk(BoardNum)
       Input #1, user_ba(BoardNum), flexfilename(BoardNum)
       Input #1, a
     Next b
   Close #1
   Exit Sub

ini_err_handler:
   Exit Sub

End Sub

Function FlexSendVB(altera() As Integer, NumPnts As Long)

Static Status As Integer
Static j As Long
Static ConfigData As Integer

Debug.Print altera(5)
Debug.Print NumPnts

'Reset Flex Chip
   MSG = write_port(FlexCtrlPort, nConfigLo, 1)

   MSG = write_port(FlexCtrlPort, nConfigHI, 1)
```

```
'Check for FLEX Proper Response
    j = 0
    MSG = write_port(FlexCtrlPort, nConfigHI_nCSLO, 1)
    MSG = read_port(FlexStatusPort, Status, 1)
    While Status <> nStatLO_RDYnBSYHI
        MSG = read_port(FlexStatusPort, Status, 1)

        j = j + 1
        If j > 200 Then
            FlexSendVB = -2
            Exit Function
        End If
    Wend

    For j = 1 To NumPnts
        ConfigData = altera(j)
        MSG = write_port(FlexDataPort, ConfigData, 1)
    Next j

    MSG = write_port(FlexCtrlPort, nConfigHI_nCSHI, 1)

    Delay (0.5) 'Wait half a second before getting status
    MSG = read_port(FlexStatusPort, Status, 1)
    Status = Status And &H1C        'And out unused bits
        If Status <> ConfDone Then      'Is Conf_Done & nStatus HI
        FlexSendVB = -3
        Exit Function
    End If

    FlexSendVB = True

End Function


Function FlexConfigVB(filename As String) As Integer

Dim line As String
Static i As Long
Static CommaLeft As Integer
Static CommaRight As Integer
Static Token As Integer

'main.flexstatus.Text = "Reading Flex File " + filename
'main.flexstatus.BackColor = LtGray
DoEvents

Open filename For Input As #1
i = 1
Do While Not EOF(BoardNum) ' Loop until
    Line Input #1, line    ' Read data into two variables.
'     Debug.Print line  ' Print data to Debug window.
    CommaLeft = 1
    CommaRight = -1
    Do While True
        CommaRight = InStr(CommaLeft, line, ",", 1)
        If CommaRight = 0 Then  'If not at first character then exit
            If CommaLeft <> 1 Then Exit Do
```

```
            CommaRight = 10
        End If
        Token = Mid(line, CommaLeft, CommaRight - CommaLeft)
        altera(i) = Val(Token)
        i = i + 1
        CommaLeft = CommaRight + 1
    Loop
Loop

Close #1    ' Close file.
MaxAlteraPnts = i - 1 'Fix total number of bytes read
FlexConfigVB = True
End Function
Function Flx10KSetAddr(Index As Integer)

    Dim MSG As Integer, CBA As Integer, UBA As Integer

    CBA = FlexCtrlPortBA(Index)
    UBA = FlexUserBA(Index)
    FlexCtrlPort = CBA                      'Control Port
    FlexDataPort = CBA + 1                  'Data Programming Port
    FlexStatusPort = CBA                    'Status Port
    FlexUserCtrlPort = (CBA And &H3F0) + 4  'Flex User Control Port (A or B)
    FlexUserBasePort = UBA

    ' RMS 17 Sep 96
    ' These do not make sense with the new (or old) U4 map.  Bob?
    DP_MEM_Right_Addr_Lo = CBA + 3
    DP_MEM_Right_Addr_HI = CBA + 5


End Function


Function FlexSendBuffer()
Dim MSG As Long

'main.flexstatus(0).Text = "Sending Data "
MSG = FlexSendVB(altera(), MaxAlteraPnts)
If MSG <> MaxAlteraPnts Then
'    main.flexstatus(0).Text = "Error Configuring Flex"
'    main.flexstatus(0).BackColor = red
    FlexSendBuffer = False
    Exit Function
End If

'main.flexstatus(0).Text = "Flex Configured"
'msg = GetFlexSDatus()

FlexSendBuffer = True
End Function

Sub InitFlx10KAddr(BoardNum As Integer)

    'Set Altera 5192 Base Addresses
        FlexCtrlPort = FlexCtrlPortBA(BoardNum) + 0
        FlexDataPort = FlexCtrlPortBA(BoardNum) + 1
```

```
        FlexStatusPort = FlexCtrlPortBA(BoardNum) + 2
        'optFlxBaseAddr(Board).Value = True

    'Set FLEX Address for Left side of Dual Port Memory
        DP_MEM_Left_Data = FlexUserBA(BoardNum) + 0
        DP_MEM_Left_Addr_Lo = FlexUserBA(BoardNum) + 1
        DP_MEM_Left_Addr_HI = FlexUserBA(BoardNum) + 2
        DP_MEM_Left_Ctrl = FlexUserBA(BoardNum) + 3

    'Set FLEX Address for Right side of Dual Port Memory
        DP_MEM_Right_Data = FlexUserBA(BoardNum) + 4
        DP_MEM_Right_Addr_Lo = FlexUserBA(BoardNum) + 5
        DP_MEM_Right_Addr_HI = FlexUserBA(BoardNum) + 6
        DP_MEM_Right_Ctrl = FlexUserBA(BoardNum) + 7

        DP_MEM_Left_addr_Mux = FlexUserBA(BoardNum) + &HC


End Sub


'Note that 'FlexUserBA' is determined by .ttf design file
'FlexCtrlPortBA(BoardNum) is the board addr. determined by wire straps to
5192
'
'This Routine returns the number of Files read
Public Sub InitBoardBase(BrdType As String)
 Dim i As Integer
 Dim dum As String
 Dim line As String
 Dim line2 As String
 Dim filename As String

'************
Exit Sub

 filename = "Win" + BrdType + ".ini"

Open filename For Input As #1

Input #1, NumFlexFiles, dum
For i = 0 To NumFlexFiles - 1
   Input #1, flexfilename(i), FlexFileDoc(i)
Next i

' Skip Three lines
Line Input #1, dum
Line Input #1, dum
Line Input #1, dum

i = 0
Do Until EOF(1)
   Input #1, FlexCtrlPortBA(i), FlexUserBA(i), dum     '5192 and FLEX base
addr
   i = i + 1 .
Loop
```

```
If (i > 8) Then
   NumFlexes = 8
   MsgBox ("File " + filename + " contains too many base addresses.")
Else
   NumFlexes = i
End If

Close #1

End Sub
Sub init_flex_param()

' 5032 or 5192 Addresses
        FlexCtrlPort = FlexCtrlPortBA(BoardNum) + 0
        FlexDataPort = FlexCtrlPortBA(BoardNum) + 1
        FlexStatusPort = FlexCtrlPortBA(BoardNum) + 2

        FlexOFFPort = FlexCtrlPortBA(BoardNum) + 6
        FlexONPort = FlexCtrlPortBA(BoardNum) + 7

        flex_user_ba_ctrl = FlexCtrlPortBA(BoardNum) + 3

End Sub



'I input parameter sets the address of the right port
'on dual port memory (0-4095)

Function SetAddrRight(i As Integer)
  Static LoAddr As Integer
  Static HiAddr As Integer

    If i > 4095 Then
        SetAddrRight = False: Exit Function
    End If

    LoAddr = i Mod 256
    HiAddr = i \ 256
    MSG = write_port(DP_MEM_Right_Addr_Lo, LoAddr, 1)
    MSG = write_port(DP_MEM_Right_Addr_HI, HiAddr, 1)

    SetAddrRight = True

End Function
Private Sub FlxBaseAddr(Index As Integer)
Dim a
Dim MSG As Integer, CBA As Integer, UBA As Integer

        'Definition of Ports used to program  and control the FLEX chip
        'on a 10K50 board

            Flx10KSetAddr (Index)
            MSG = Get10KStatus()

'        optFlxBaseAddr(Index).Value = True
```

```
        FlexIndex = Index

'
        'Get and Display Status of Current Flex Chip
        MSG = Get10KStatus()

End Sub


'I input parameter sets the address of the left port
'on dual port memory (0-4095)

Function SetAddrLeft(i As Integer)
  Static LoAddr As Integer
  Static HiAddr As Integer

    If i > 4095 Then
        SetAddrLeft = False
        Exit Function
    End If

    LoAddr = i Mod 256
    HiAddr = i \ 256        'Be Sure to Integer divide
    MSG = write_port(DP_MEM_Left_Addr_Lo, LoAddr, 1)
    MSG = write_port(DP_MEM_Left_Addr_HI, HiAddr, 1)

    SetAddrLeft = True

End Function


Function usecDelay(Dwell As Integer)
Dim initClick As Long
Dim currentClick As Long
Dim EndClick As Long
Dim icnt As Long
Dim ret As Integer

EndClick = Dwell / 0.8381    'Each click represents 0.8381 usec

ret = InitClicks
initClick = Clicks          'Sets Down counter to max value ( about 65,000)
If initClick < 0 Then
   initClick = 65535 + initClick
End If
currentClick = Clicks       'Reads current count
If currentClick < 0 Then
   currentClick = 65535 + currentClick
End If

icnt = 0

While (EndClick > (initClick - currentClick))
    currentClick = Clicks
    If currentClick < 0 Then
       currentClick = 65535 + currentClick
    End If
```

```
    icnt = icnt + 1
    If icnt > 1000000 Then
        usecDelay = False
        Exit Function
    End If
Wend

usecDelay = True
End Function


Sub Board_Bit()
Dim i As Integer
  '  init_flex_param
    flexform.ini_num.Text = NumBds
    num_bds_fnd = 0

'** find # of boards that respond to ping **
For BoardNum = 1 To NumBds
    Flx10KSetAddr (BoardNum)
        flexform.addr(BoardNum).Text = Hex(FlexCtrlPortBA(BoardNum))
    ck_bd_present
    If present(BoardNum) = False Then
        flexform.present(BoardNum).BackColor = red
    Else
        flexform.present(BoardNum).BackColor = green
        num_bds_fnd = num_bds_fnd + 1
    End If
Next BoardNum

    flexform.found_num.Text = num_bds_fnd

End Sub
Sub ck_bd_present()
Static MSG As Integer

    MSG = write_port(FlexUserCtrlPort, &H1, 1)      'Turn on Read Buffer
Capability

    MSG = read_port(FlexStatusPort, fstatus(BoardNum), 1)        'Get Flex
Status

    If fstatus(BoardNum) = &HFF Then
      present(BoardNum) = False
    Else
      present(BoardNum) = True
    End If

End Sub
Sub test_boards()
Dim dum As Integer, dly As Long, invar As Integer

    flexform.Show
'** set initial state to gray in leds
    For dum = 1 To NumBds
        flexform.present(dum).BackColor = LtGray
        flexform.Bstatus(dum).BackColor = LtGray
```

```
        flexform.Bconfigdone(dum).BackColor = LtGray
        flexform.BBIT(dum).BackColor = LtGray
    Next dum
'** check presence of boards ***
    get_flex_ini     ' read basic flex addrs. & pri param. from filename$.ini
file
    reset_flexs

    Board_Bit        ' check # of boards and operational status

'** load flex chips & display status **
  For BoardNum = 1 To NumBds
   If present(BoardNum) = True Then
        InitFlexType
        flexform.Bconfigdone(BoardNum).BackColor = yellow
        MSG = LoadFlex(flexfilename(BoardNum), 1)
        Flx10KSetAddr (BoardNum)
        Get10KStatus
     If Status(BoardNum) = False Then
        flexform.Bstatus(BoardNum).BackColor = red
     Else
        flexform.Bstatus(BoardNum).BackColor = green
     End If
'*** write user base addr. to flex (a=3x6, b=3x7)
     offset = (0.5 * (FlexCtrlPortBA(BoardNum) And &H2) + 6) -
(FlexCtrlPortBA(BoardNum) And &H2)
     dum = write_port(FlexCtrlPortBA(BoardNum) + offset, user_ba(BoardNum) \
&H4, 1) 'set Flex User BaseAddr
      If configdone(BoardNum) = False Then
        flexform.Bconfigdone(BoardNum).BackColor = red
      Else
        flexform.Bconfigdone(BoardNum).BackColor = green
      End If
        flexform.filename(BoardNum).Text = flexfilename(BoardNum)
        flexform.userBA(BoardNum).Text = Hex(user_ba(BoardNum))
'** check BIT register for proper load
        dum = write_data(&HFC, BoardNum * &H15A5, 2) 'write board (internal
addr (linear addr. 0 ->255 words) , data , word)
        dum = read_data(&HFC, invar, 2)
         If invar = BoardNum * &H15A5 Then
            flexform.BBIT(BoardNum).BackColor = green
         Else
            flexform.BBIT(BoardNum).BackColor = red
         End If
'        dum = write_port(user_ba(BoardNum), 0, 2)    ' zero BIT reg. for
noninterference w/next board
   End If   'end check for board present
  Next BoardNum

      For dly = 1 To 10000: DoEvents: Next dly
End Sub
Sub reset_flexs()   'reset flex chips (unload)
Dim dum As Integer

  get_flex_ini
  For BoardNum = 1 To NumBds
```

```
    MSG = write_port(FlexCtrlPortBA(BoardNum), nConfigLo, 1)   'Set nConfig
(bit 0) LO
'     MSG = write_port(FlexCtrlPortBA(BoardNum), nConfigHI, 1)   'Set nConfig
(bit 0) LO
  Next BoardNum

'** set initial state to gray in leds
    For dum = 1 To NumBds
        flexform.present(dum).BackColor = LtGray
        flexform.Bstatus(dum).BackColor = red
        flexform.Bconfigdone(dum).BackColor = LtGray
        flexform.BBIT(dum).BackColor = LtGray
    Next dum

    Board_Bit

End Sub
Function write_data(iaddr As Integer, idata As Integer, iword As Integer) As
Integer
    'MSG = write_port(user_ba(BoardNum) + 2, iaddr, 2) 'internal addr of I/O
- 256 addrs. per Flex chip
    'MSG = write_port(user_ba(BoardNum) + 0, idata, 2) 'internal data of I/O
- always 16 bit (word) write
    MSG = write_port(FlexUserCtrlPort, &H0, 1)
    MSG = write_port(user_ba(&H1) + 2, iaddr, 2) 'internal addr of I/O - 256
addrs. per Flex chip
    MSG = write_port(user_ba(&H1) + 0, idata, 2) 'internal data of I/O -
always 16 bit (word) write
End Function
Function read_data(iaddr As Integer, idata As Integer, iword As Integer) As
Integer
    MSG = write_port(FlexUserCtrlPort, &H0, 1)
    MSG = write_port(user_ba(&H1) + 2, iaddr, 1) 'internal addr of I/O - 256
addrs. per Flex chip
    MSG = write_port(FlexUserCtrlPort, &H1, 1)
    MSG = read_port(user_ba(&H1) + 0, idata, 2) 'internal data of I/O -
always 16 bit (word) write
End Function
```

global (Global.bas)

```
Global crystal_clk(8) As Single

'***** color definitions ****
Global Const red = &HFF&
Global Const blue = &HFF0000
Global Const green = &HFF00&
Global Const black = &H0
Global Const yellow = &HFFFF&
Global Const brown = &H80FF&
Global Const ltblue = &HFFFF00
Global Const LtGray = &H8000000F
Global Const DkGray = &H808080
Global Const Beige = &HC0FFFF
```

MainMod (main.bas)

```
Option Explicit

Sub Main()
Dim dum As Integer, cnt As Integer, invar As Integer

'*** open running windows ***
    flexform.Show   'flexform.frm

'*** check the latch values ***
    test_boards ' flexfunc.bas
     readPara     ' file.bas
     readRaw      ' file.bas
     readCosine   ' file.bas
     readSine     ' file.bas
     isar         ' the_isar.bas
     ' pep_test          'peptest.bas'
End Sub
```

the_isar (the_isar.bas)

```
Option Explicit
'***** internal addresses for the ISAR program *****
Global Const phiAddr = &H10          ' for doppler offset
Global Const gainAddr = &H20         ' for gain
Global Const tapAddr = &H30          ' for tap delay line
Global Const modPulseAddr = &H40     ' for modulated pulse readback
Global Const feedback = &H60          ' for reading back values

Public Sub isar()

Dim batchCnt As Integer
Dim pulseCnt As Integer
Dim intraPulseCnt As Integer
Dim tapCnt As Integer
Dim Tap(3) As Integer
Dim Phase(3) As Integer
Dim rgain(3) As Integer
Dim GainOutI(3) As Integer
Dim GainOutQ(3) As Integer
Dim Acc(3) As Integer
Dim Lut(6) As Integer
Dim finalAcc(4) As Integer
Dim tmp As Integer
Dim dummy1, dummy2 As Double
Dim LutI As Double
Dim LutQ As Double
Dim GainOutIdec, GainOutQdec As Double
Dim dummy3, dummy4 As Double
Dim idx As Integer
Dim sumI, sumQ As Double
Dim finalAccI, finalAccQ As Double
Dim finalAccI_NEW, finalAccQ_NEW As Double 'for test of Public Function
numFormConv
Dim number As Integer   'for test of Public Function twoComplement2Float
Dim test As Double      'for test of Public Function twoComplement2Float
Dim n As Integer        'for test of Public Function twoComplement2Float

'load data files (file.bas)
readPara
readRaw

'Open file for write
Open "lets_check.txt" For Output As #1
Open "imagei.txt" For Output As #2
Open "imageq.txt" For Output As #3

'test of Public Function twoComplement2Float
'Print #1, "Test of Public Function twoComplement2Float: "
'n = 4 'number of bits
'For number = 0 To (2 ^ n - 1)
'    test = twoComplement2Float(number, n) '* 2 ^ (n - 1)
'    Print #1, number, "=", test 'print to lets_check
'Next number
```

```
'Initialize gain values
flexform.TGain(0).Text = Gain(0)
flexform.TGain(1).Text = Gain(1)
flexform.TGain(2).Text = Gain(2)

'Reset tap-delay line
MSG = write_data(tapAddr, 0, 2)
MSG = write_data(tapAddr, 1, 2)

'loop for batch
For batchCnt = 0 To nDopplerCell - 1
    '
    'loop for intra-pulse: repeat for number of range gates + target extent
    For intraPulseCnt = 0 To (nRangeCell + targetExtent - 1)
        '-----------------------------------------------------------------
        'Write Phi and Gain values
        For tapCnt = 0 To targetExtent - 1
            '
            'load doppler offset parameters
            'MSG = write_data(phiAddr + tapCnt, Phi(nDopplerCell - batchCnt -
1, tapCnt), 2)
            MSG = write_data(phiAddr + tapCnt, Phi(batchCnt, tapCnt), 2)
            '
            Print #1, "batchCnt=", batchCnt
            Print #1, "intraPulseCnt=", intraPulseCnt
            Print #1, "tapCnt=", tapCnt
            Print #1, "Phi(batchCnt, tapCnt)=", Phi(batchCnt, tapCnt)
            '
            'load gain parameters
            MSG = write_data(gainAddr + tapCnt, Gain(tapCnt), 2)
            '
        Next tapCnt
        '
        '-----------------------------------------------------------------
        'Read back gain values
        For idx = 0 To 2
            MSG = read_data(gainAddr + idx, rgain(idx), 2)
            rgain(idx) = rgain(idx) And &H7
            flexform.TGain(idx).Text = rgain(idx)
            Print #1, "rgain=", rgain(idx)
        Next idx
        '-----------------------------------------------------------------
        'Read back phi values
        For idx = 0 To 2
            MSG = read_data(phiAddr + idx, Phase(idx), 2)
            Phase(idx) = Phase(idx) And &H1F
            flexform.TPhi(idx).Text = Phase(idx)
            Print #1, "Phase=", Phase(idx)
            Print #1, "Phase(idx)=", twoComplement2Float(Phase(idx), 5) * (2
^ 7), "converted from 2-complement"

        Next idx
        '-----------------------------------------------------------------
        ' Strobe into delay line
        MSG = write_data(tapAddr + 1, 0, 2) ' ripple data
        MSG = write_data(tapAddr + 2, DRFM(batchCnt, intraPulseCnt), 2) '
Strobe in new data
```

```
        Print #1, "DRFM(batchCnt, intraPulseCnt)=", DRFM(batchCnt,
intraPulseCnt)
        Print #1, "DRFM(batchCnt, intraPulseCnt)=",
twoComplement2Float(DRFM(batchCnt, intraPulseCnt), 5) * (2 ^ 7), "converted
from 2-complement"
        '
        'Read back tap values
        For idx = 0 To 2
            MSG = read_data(tapAddr + idx, Tap(idx), 2)
            Tap(idx) = (Tap(idx) And &H1F)
            flexform.TTap(idx).Text = Tap(idx)
            Print #1, "Tap=", Tap(idx)
        Next idx
        '-----------------------------------------------------------------
        'Read back ph_acc values (mod 32)
        For idx = 0 To 2
            MSG = write_data(feedback, idx, 2)
            MSG = read_data(feedback, Acc(idx), 2)
            Acc(idx) = Acc(idx) And &H1F
            flexform.TAcc(idx).Text = Acc(idx)
            Print #1, "Acc=", Acc(idx)
            Print #1, "Acc(idx)=", twoComplement2Float(Acc(idx), 5) * (2 ^
7), "converted from 2-complement"

        Next idx
        '-----------------------------------------------------------------
        'Read back LUT values
        tmp = 3
        'sumI = 0
        'sumQ = 0
        For idx = 0 To 2
            MSG = write_data(feedback, tmp + (idx * 2), 2)
            MSG = read_data(feedback, Lut(idx * 2), 2)
            MSG = write_data(feedback, tmp + (idx * 2 + 1), 2)
            MSG = read_data(feedback, Lut(idx * 2 + 1), 2)
            Lut(idx * 2 + 1) = (Lut(idx * 2 + 1) And &HFF)
            Lut(idx * 2) = (Lut(idx * 2) And &HFF)
            dummy1 = Val(Format(twoComplement2Float(Lut(idx * 2), 8),
"##.###"))
            dummy2 = Val(Format(twoComplement2Float(Lut(idx * 2 + 1), 8),
"##.###"))
            LutI = twoComplement2Float(Lut(idx * 2), 8)
            LutQ = twoComplement2Float(Lut(idx * 2 + 1), 8)
            flexform.TLut(idx).Text = Str(dummy1) & "," & Str(dummy2)
            'Print #1, "LUT(idx*2)=", Lut(idx * 2), "dummy1=", dummy1,
"LutI=", LutI
            'Print #1, "LUT(idx*2+1)=", Lut(idx * 2 + 1), "dummy2=", dummy2,
"LutQ=", LutQ
            'Yeo's output of intermediate results
            flexform.TPhi(idx).Text = Hex(Lut(idx * 2)) & "," & Hex(Lut(idx *
2 + 1))
            'sumI = sumI + twoComplement2Float(Lut(idx * 2), 8) * Gain(idx)
            'sumQ = sumQ + twoComplement2Float(Lut(idx * 2 + 1), 8) *
Gain(idx)
            Print #1, "idx =", idx
            Print #1, "LutI("; idx; ")=", " ", Lut(idx * 2), "LutI=", LutI
```

```
            Print #1, "LutQ("; idx; ")=", " ", Lut(idx * 2 + 1), "LutQ=",
LutQ
            Print #1, "Gain("; idx; ")=", " ", Gain(idx)
        Next idx
        flexform.TSum(0).Text = Str(sumI)
        flexform.TSum(1).Text = Str(sumQ)
        flexform.TSum(2).Text = Str(0)
        flexform.TSum(3).Text = Str(0)
        'Print #2, sumI
        'Print #3, sumQ
        '-----------------------------------------------------------------
        'Read back gain block outputs (I channel - 11 bits)
        tmp = 13
        For idx = 0 To 2
            MSG = write_data(feedback, (tmp + idx), 2)
            MSG = read_data(feedback, GainOutI(idx), 2)
            GainOutI(idx) = GainOutI(idx) And &H7FF
            'GainOutIdec = Val(Format(twoComplement2Float(GainOutI(idx), 11),
"##.###"))
            GainOutIdec = twoComplement2Float(GainOutI(idx), 11)
            Print #1, "GainOutI("; idx; ")=", GainOutI(idx), "GainOutIdec=",
GainOutIdec
        Next idx
        '
        'Read back gain block outputs (Q channel - 11 bits)
        tmp = 16
        For idx = 0 To 2
            MSG = write_data(feedback, (tmp + idx), 2)
            MSG = read_data(feedback, GainOutQ(idx), 2)
            GainOutQ(idx) = GainOutQ(idx) And &H7FF
            'GainOutQdec = Val(Format(twoComplement2Float(GainOutQ(idx), 11),
"###.###"))
            GainOutQdec = twoComplement2Float(GainOutQ(idx), 11)
            Print #1, "GainOutQ("; idx; ")=", GainOutQ(idx), "GainOutQdec=",
GainOutQdec
        Next idx
        '-----------------------------------------------------------------

        DoEvents
        '-----------------------------------------------------------------
        'Read back sum values (modified code by Stig, 2 Aug -99)
        tmp = 9
        finalAccI = 0
        finalAccQ = 0
        '
        'Read back sum values (I channel - 13 bits)
        MSG = write_data(feedback, tmp + 0, 2)
        MSG = read_data(feedback, finalAcc(0), 2)
        finalAcc(0) = finalAcc(0) And &H1FFF
        dummy3 = Val(Format(twoComplement2Float(finalAcc(0), 13),
"####.###"))
        flexform.TSum(0).Text = Str(dummy3)
        finalAccI = twoComplement2Float(finalAcc(0), 13)
        finalAccI_NEW = numFormConv(finalAcc(0))
        flexform.TSum(0).Text = Str(finalAccI)
        Print #1, "finalAcc(0)=", " ", finalAcc(0), "finalAccI = ", finalAccI
```

```
        Print #1, " - To test the new numFormConv function", "finalAccIN = ",
finalAccI_NEW
        Print #2, finalAccI
        '
        'Read back sum values (Q channel - 13 bits)
        MSG = write_data(feedback, tmp + 1, 2)
        MSG = read_data(feedback, finalAcc(1), 2)
        'Print #1, "TTTEEESSSTTT  finalAcc(1) =  ", finalAcc(1)
        finalAcc(1) = finalAcc(1) And &H1FFF
        dummy4 = Val(Format(twoComplement2Float(finalAcc(1), 13),
"####.###"))
        flexform.TSum(1).Text = Str(dummy4)
        finalAccQ = twoComplement2Float(finalAcc(1), 13)
        finalAccQ_NEW = numFormConv(finalAcc(1))
        flexform.TSum(1).Text = Str(finalAccQ)
        Print #1, "finalAcc(1)=", " ", finalAcc(1), "finalAccQ=", finalAccQ
        Print #1, " - To test the new numFormConv function", "finalAccQN = ",
finalAccQ_NEW
        Print #1, "-------------------------------------------------------------
-----------"
        Print #3, finalAccQ
        '
        flexform.TSum(2).Text = Str(0)
        flexform.TSum(3).Text = Str(0)
        '
        'tmp = 9
        'For idx = 0 To 1
        '    MSG = write_data(feedback, tmp + idx, 2)
        '    MSG = read_data(feedback, finalAcc(idx), 2)
        '    finalAcc(idx) = finalAcc(idx) And &H1FFF
        '    dummy3 = Val(Format(twoComplement2Float(finalAcc(idx), 13),
"##.###"))
        '    Print #1, "FINALACC(idx)=", finalAcc(idx), "dummy3=", dummy3
        '    flexform.TSum(idx).Text = Str(dummy3)
        '    finalAc = sumQ + twoComplement2Float(GainOutQ(idx), 11)
        'Next idx
        'Print #2, finalAcc(0)
        'Print #3, finalAcc(1)
        '
        flexform.Clock.Text = Str(batchCnt) + "," + Str(intraPulseCnt + 1)
    Next intraPulseCnt    'end intra-pulse loop
Next batchCnt    'end batch loop
Close #1
Close #2
Close #3
End Sub
Public Function twoComplement2Float(a As Integer, nbits As Integer) As Double
'modified by Stig Ekestorm, 4 Aug -99
Dim dummy1, dummy2, dummy3, dummy4 As Integer

dummy1 = 2 ^ (nbits - 1)
dummy4 = 2 ^ nbits
dummy2 = dummy1 - 1
If (a >= dummy1) Then      ' negative number test
    dummy3 = dummy1 - (a And dummy2)
    twoComplement2Float = -1 * dummy3 / (2 ^ 7) '/ dummy1 '(divide by 128 to
put the decimal point at the right position)
```

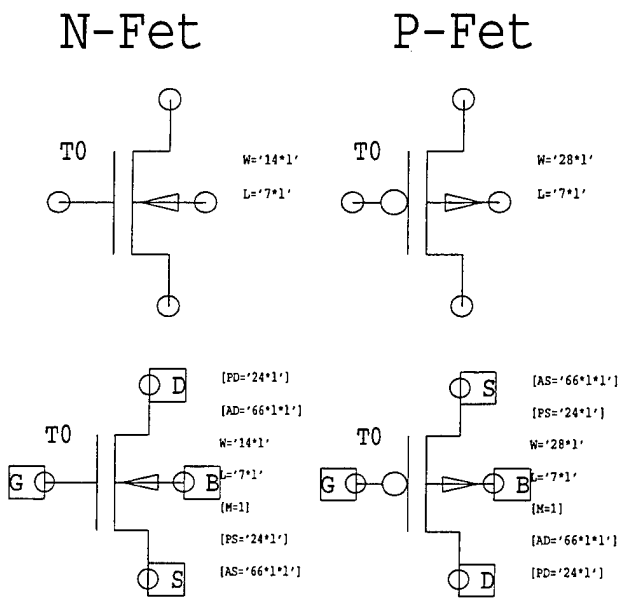
```
Else
    twoComplement2Float = a / (2 ^ 7) '/ dummy1 '(divide by 128 to put the
decimal point at the right position)
End If

If a >= 2 ^ nbits Then
    twoComplement2Float = -1111
End If
End Function
Public Function numFormConv(a As Integer) As Double 'created by Prof. Fouts,
4 Aug -99
Dim tempvar As Integer

tempvar = &H1FFF And a
If (tempvar >= 4096) Then       ' negative number test
    tempvar = tempvar Xor &H1FFF
    tempvar = tempvar + 1
    numFormConv = -1 * tempvar / (2 ^ 7)
Else
    numFormConv = tempvar / (2 ^ 7)
End If
End Function
```

Appendix C – Schematics and Symbols of Modified Architecture

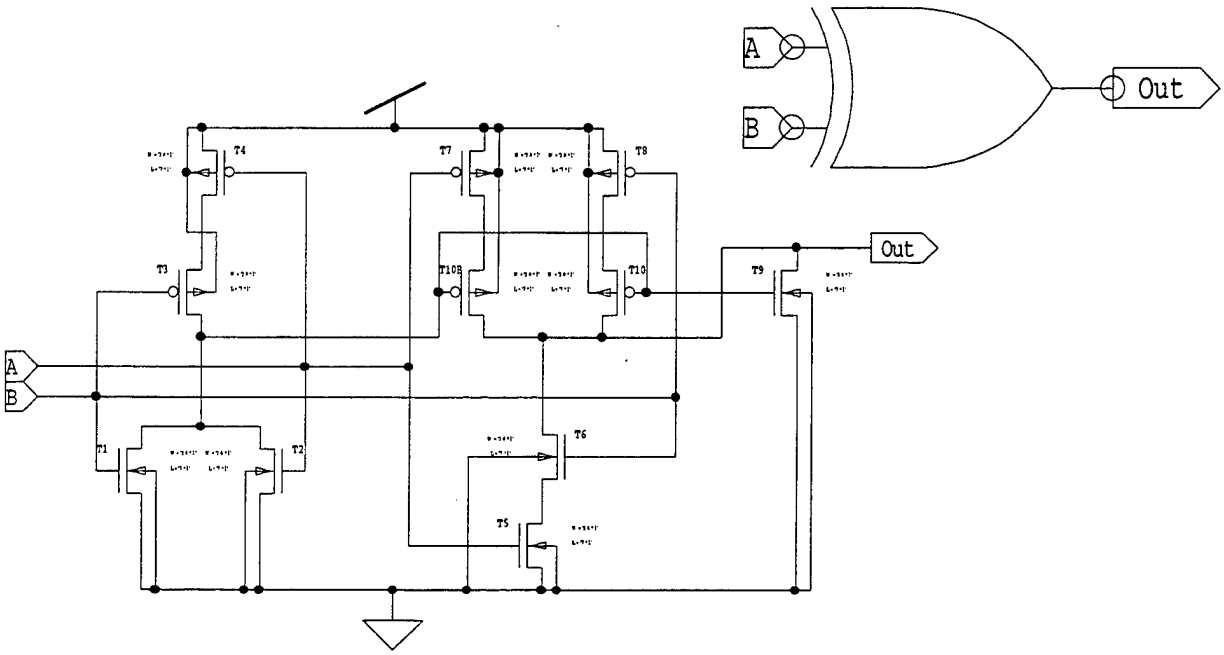
Design Building Blocks and Elements

This Appendix lists all circuits created in S-Edit. They are shown in schematic and symbol representation. The Tanner library elements are not listed and can be found in the Tanner tools Library manual.

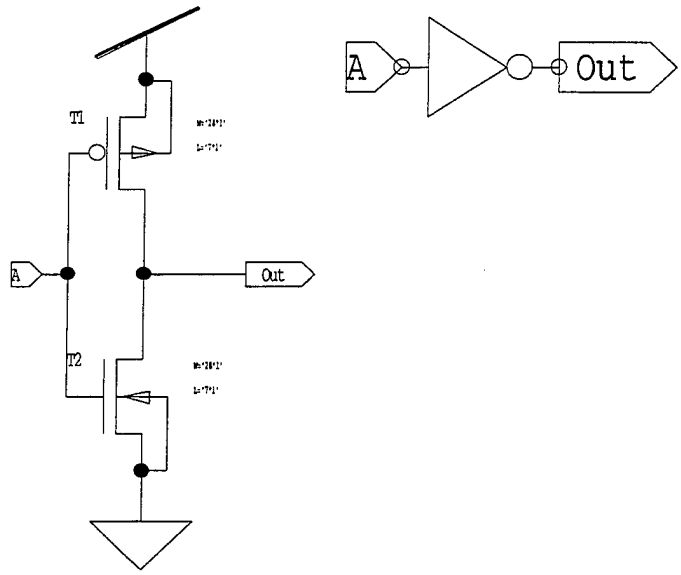
Level 1
P-Fet and N-Fet transistor



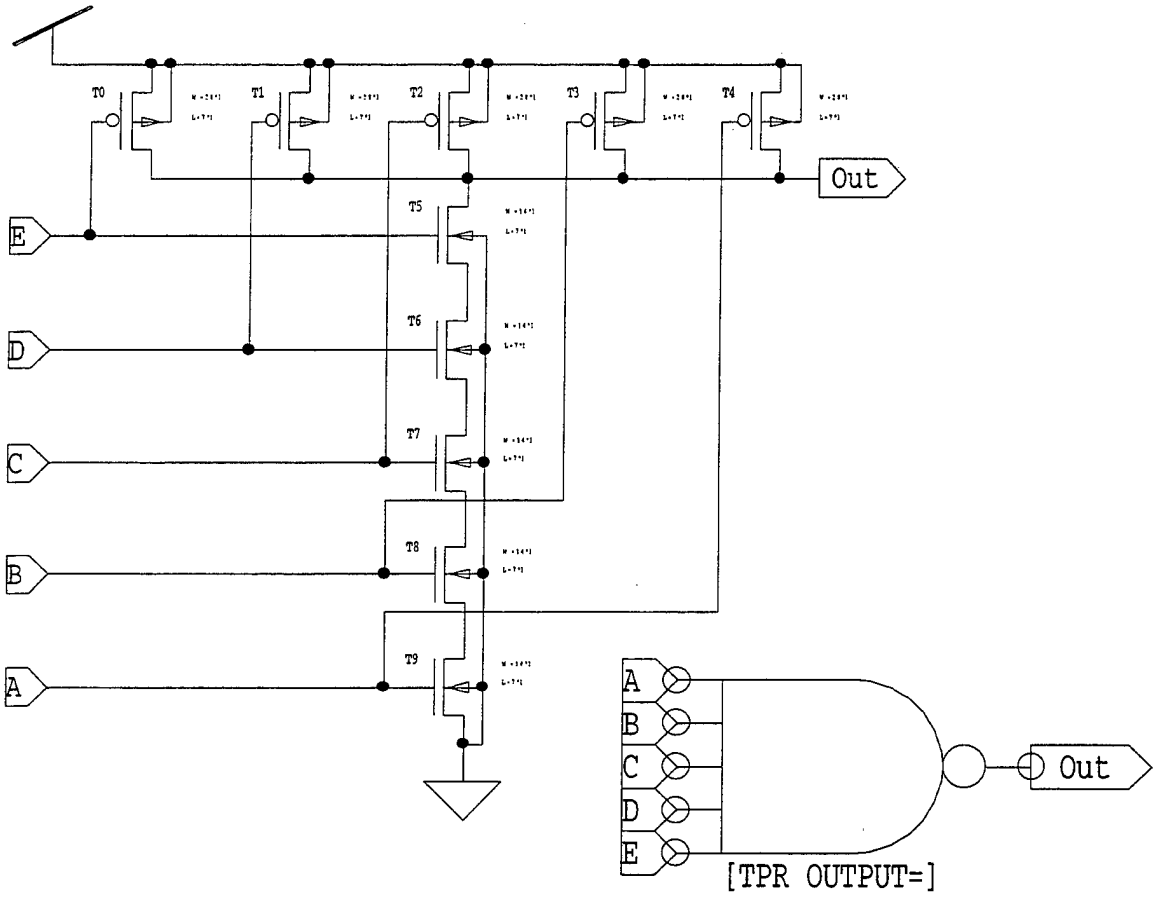
2 input XOR



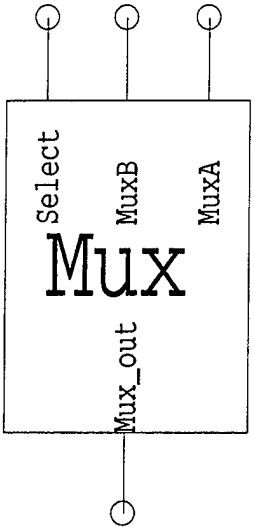
Inverter

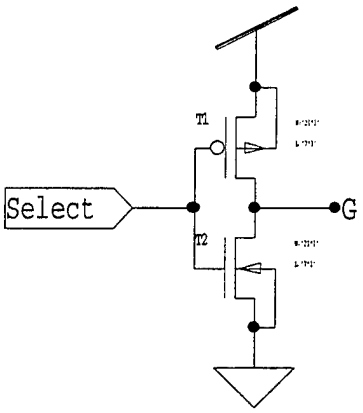


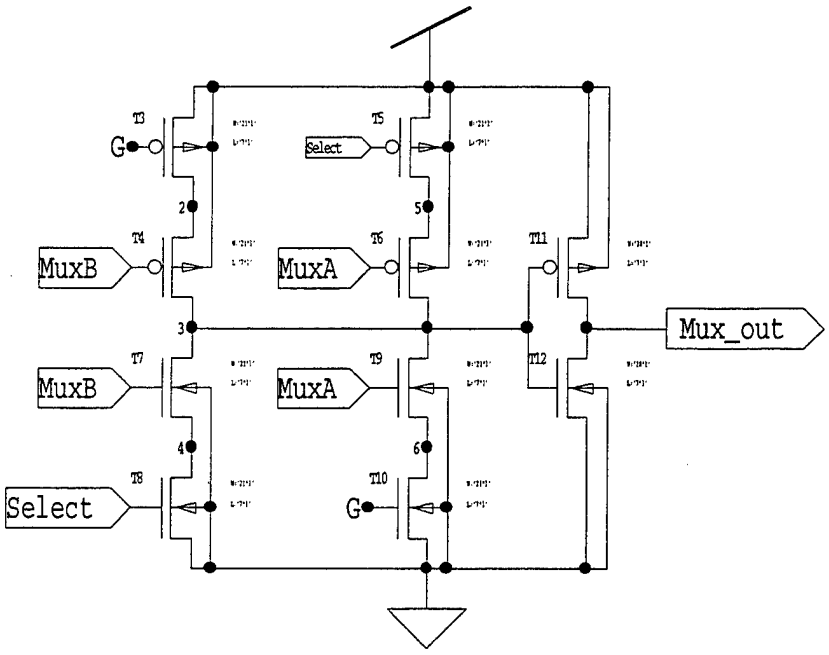
5 input NAND



Mux2

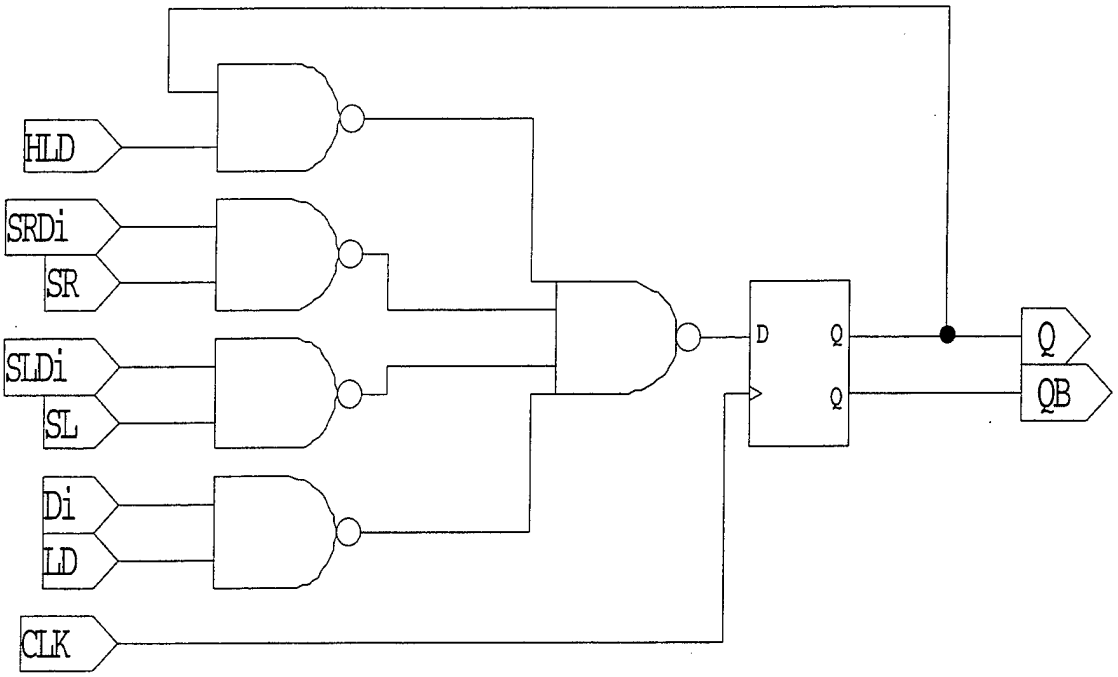


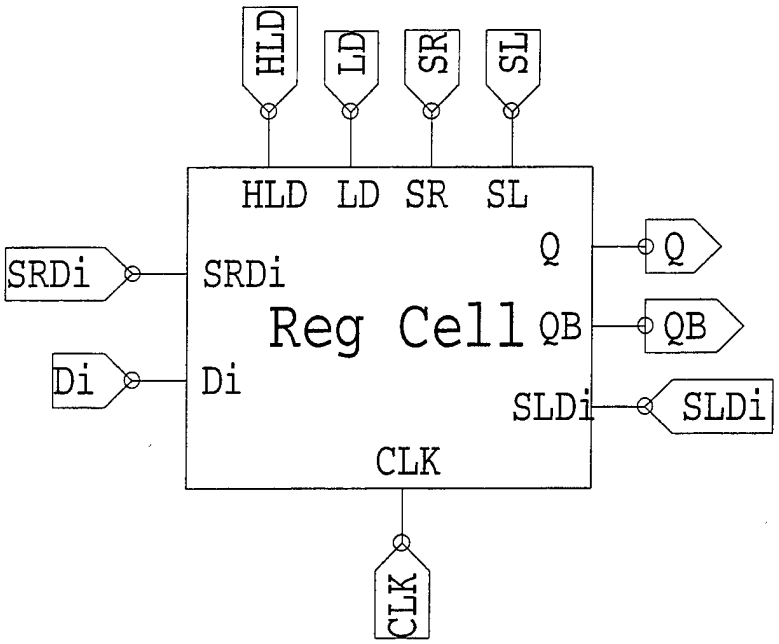




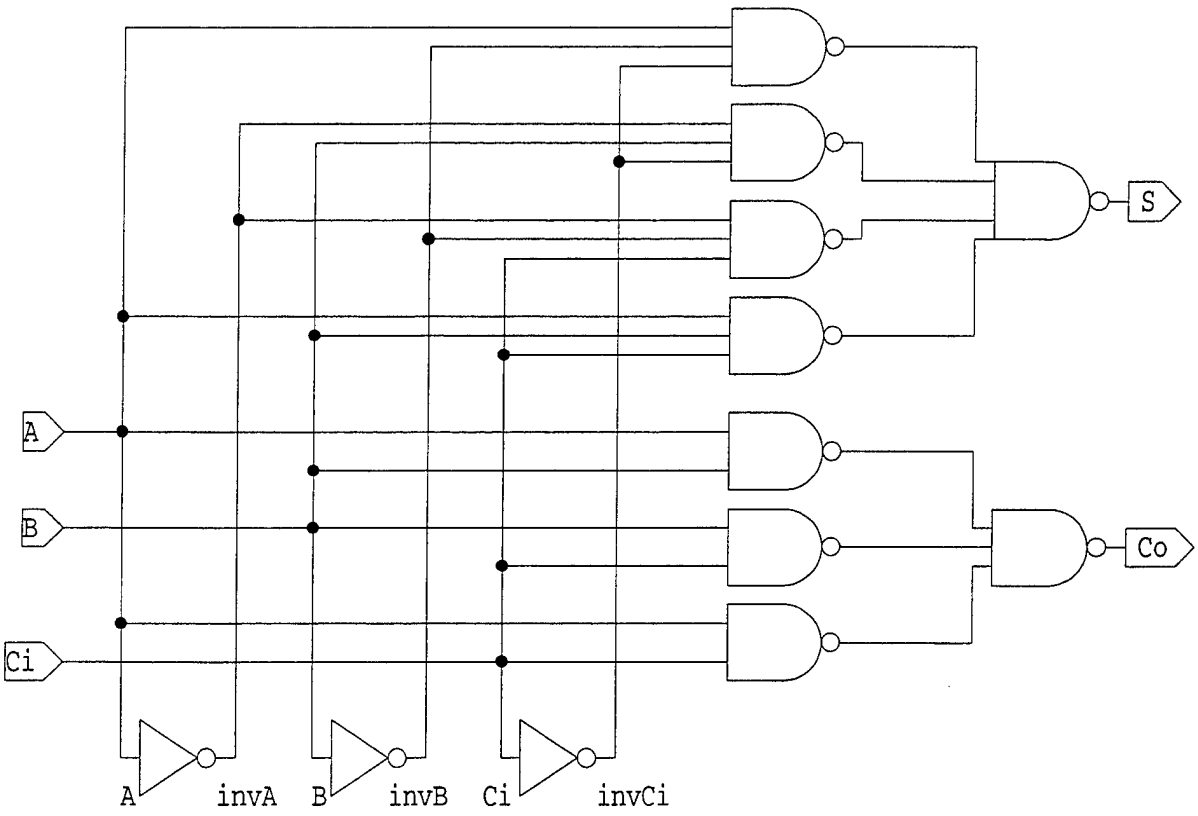
Logic Equation = (MuxA*notSelect + MuxB*Select)

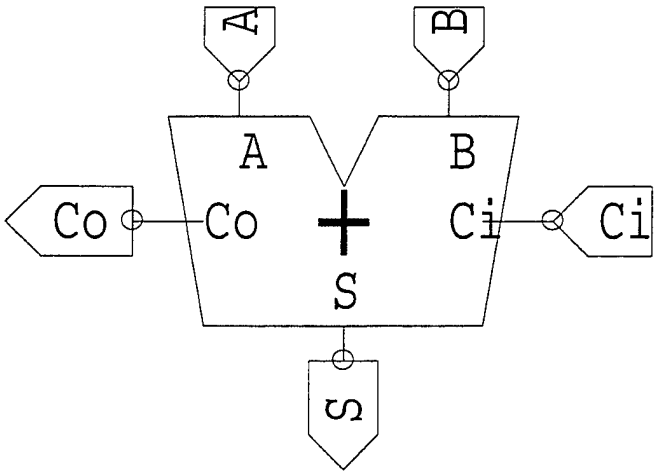
Register Cell



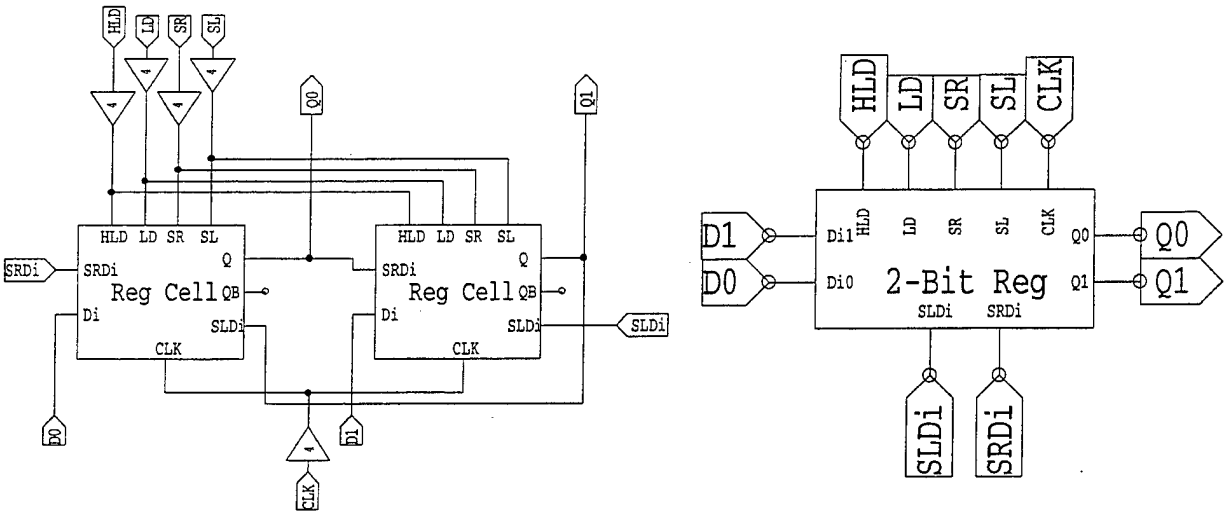


Adder Cell

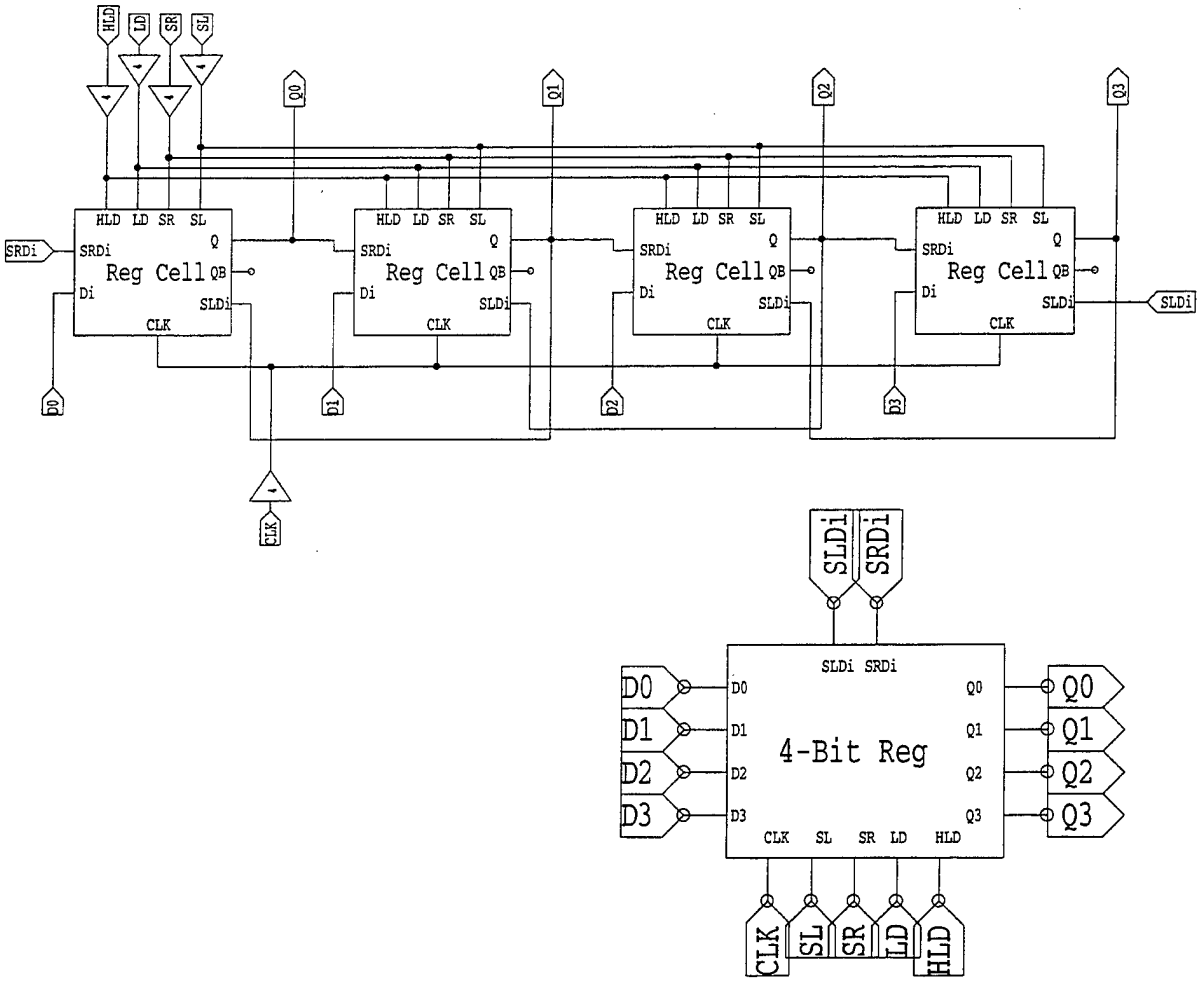




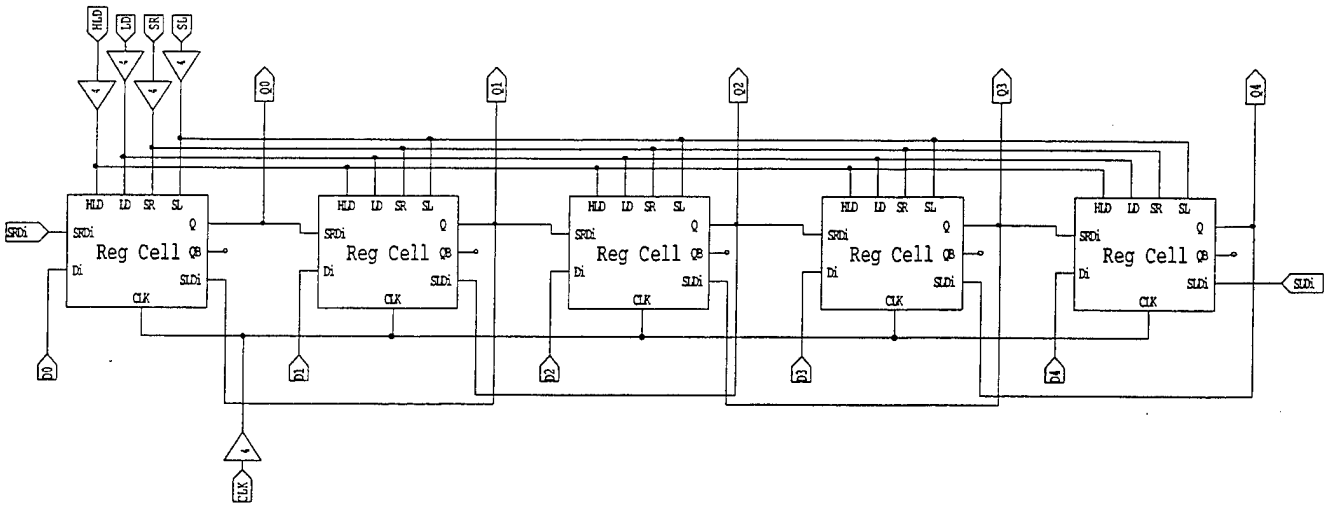
Level 2
2-bit Register

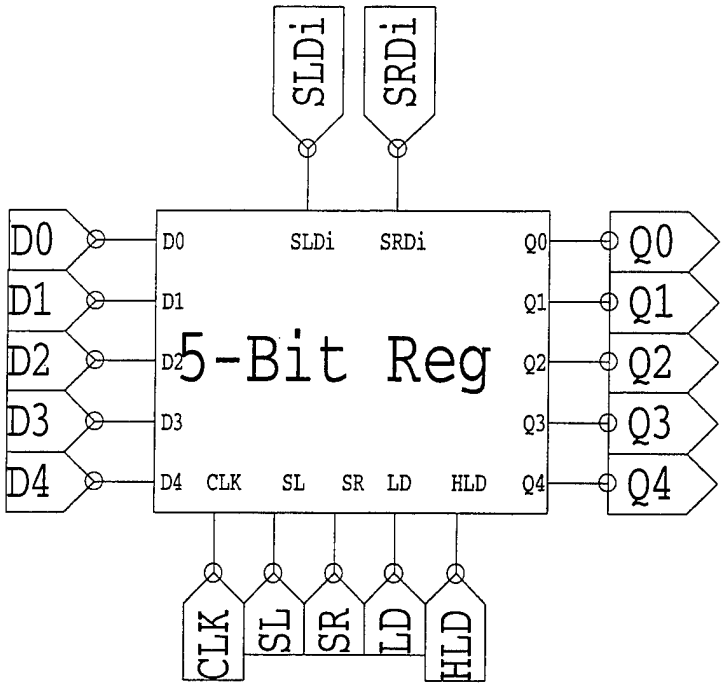


4-bit Register

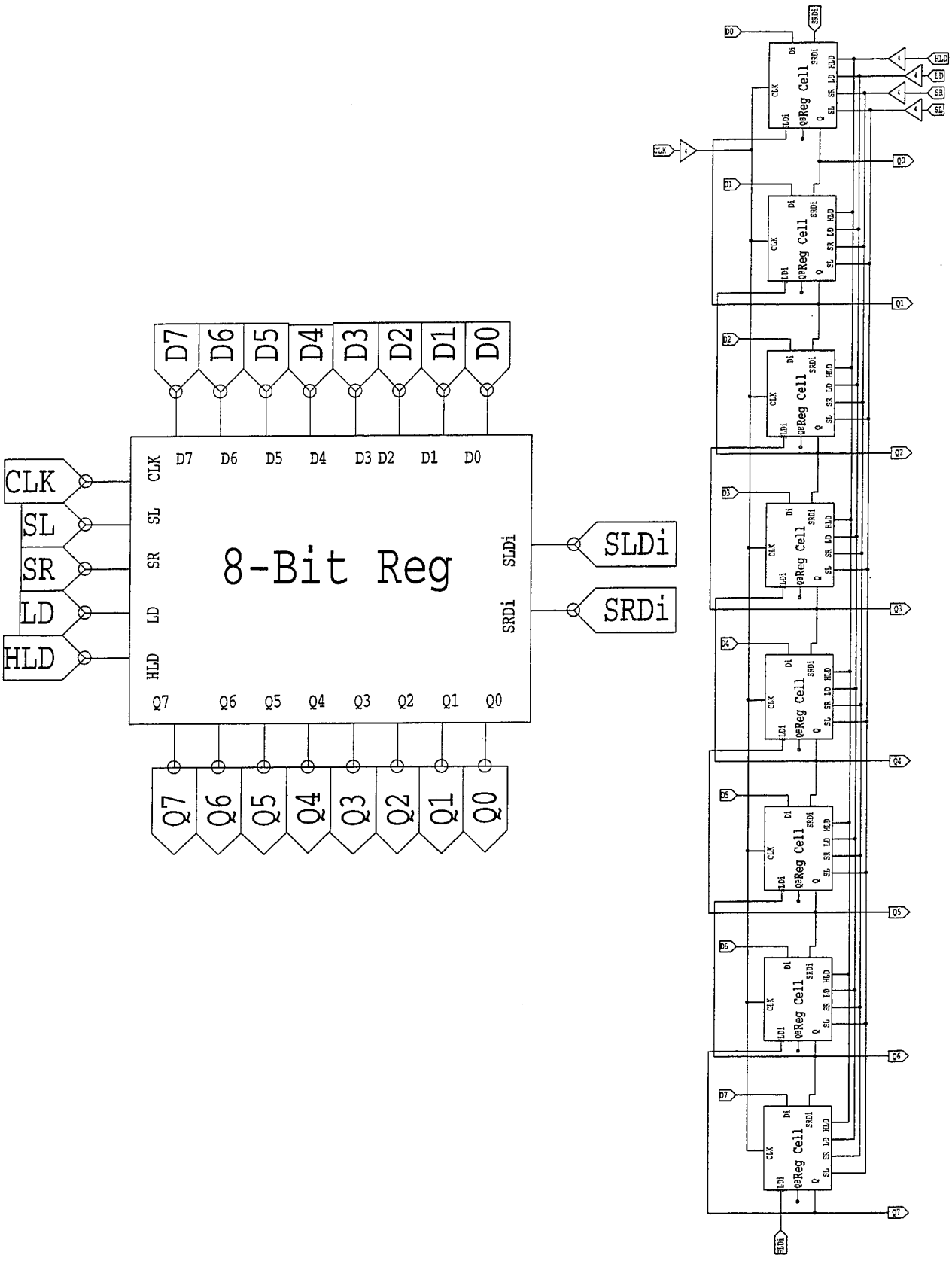


5-bit Register

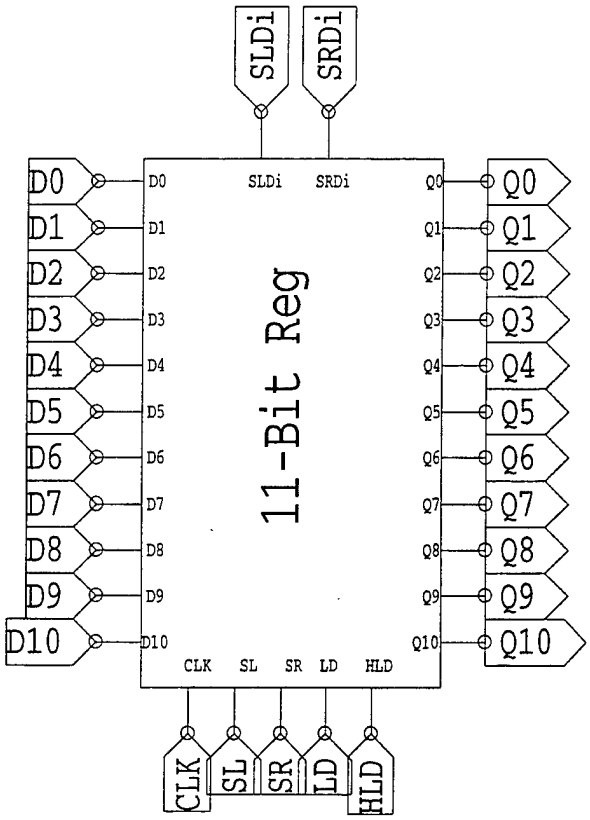


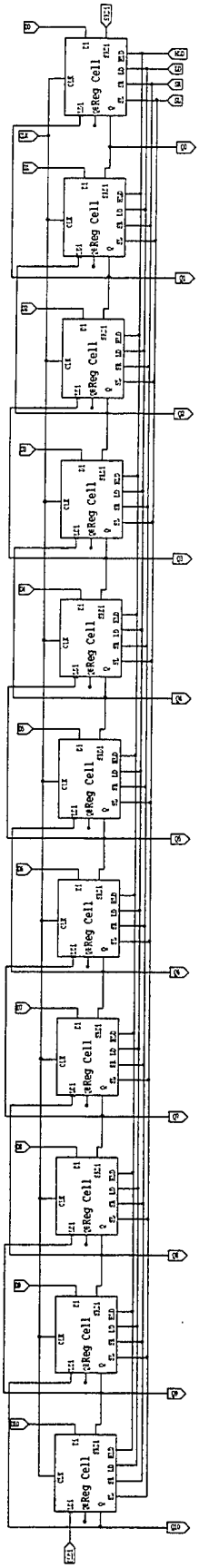


8-bit Register

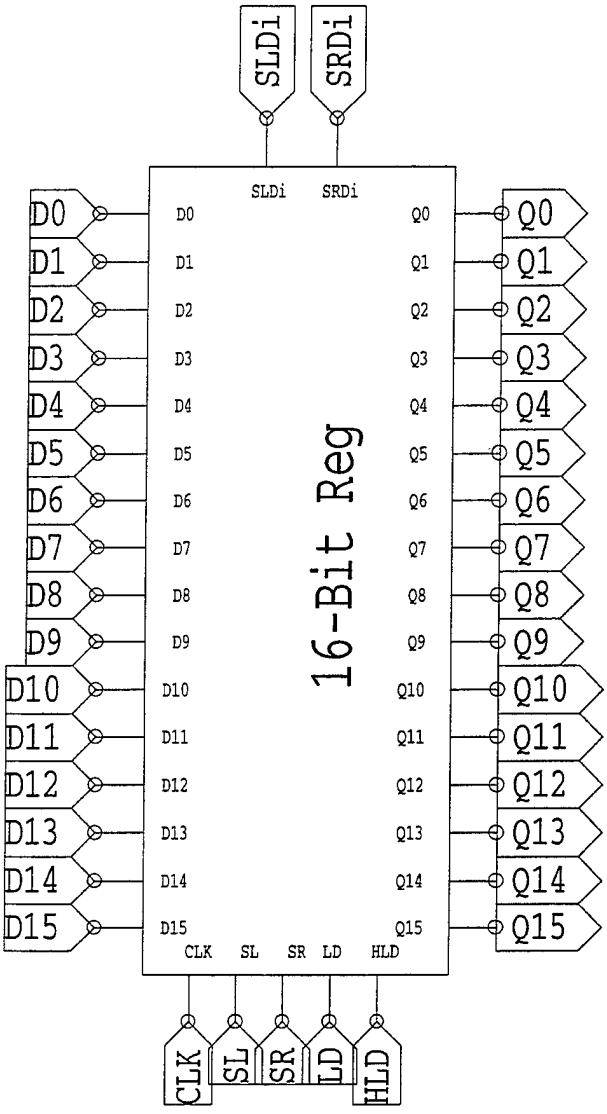


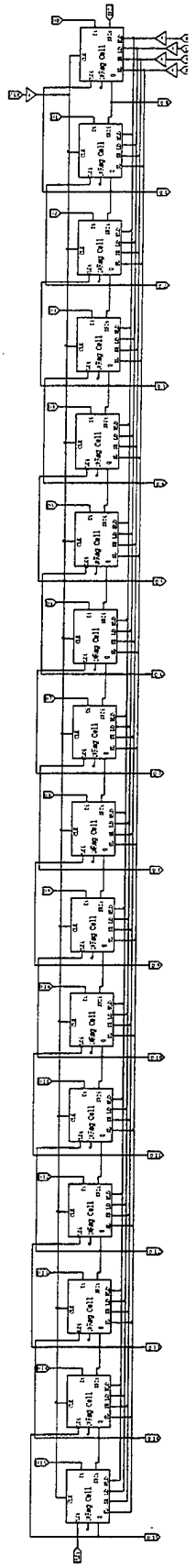
11-bit Register



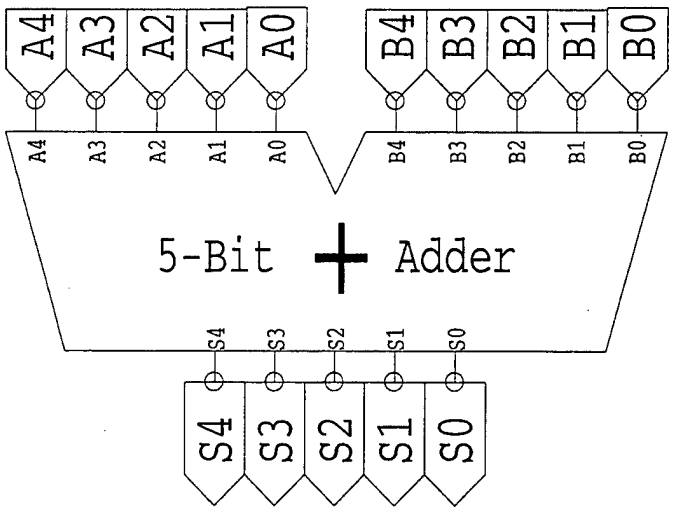


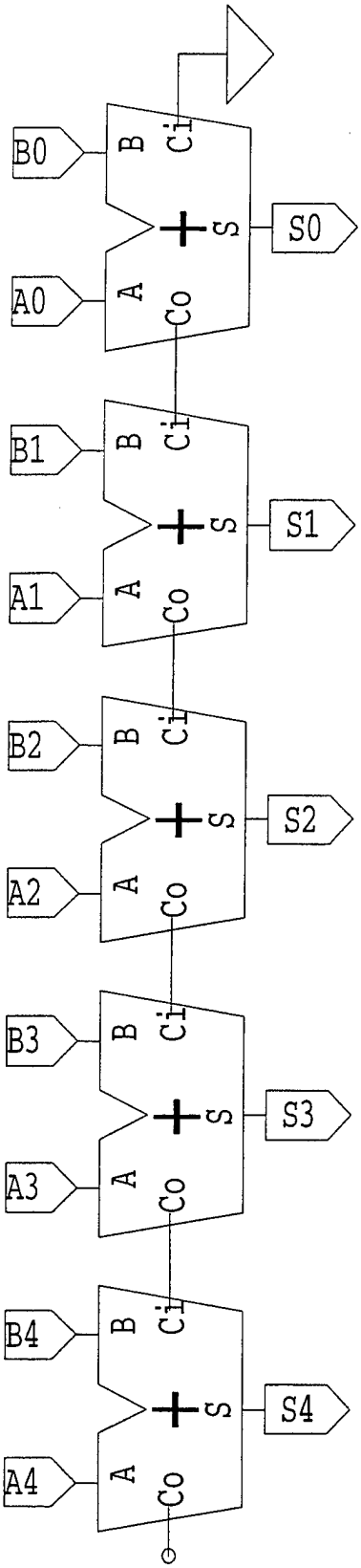
16-bit Register



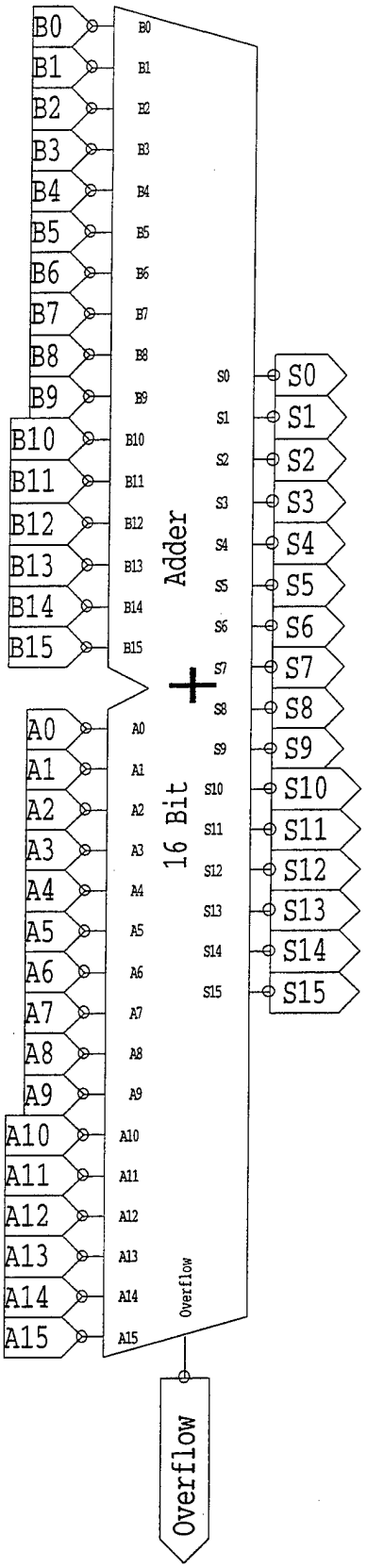


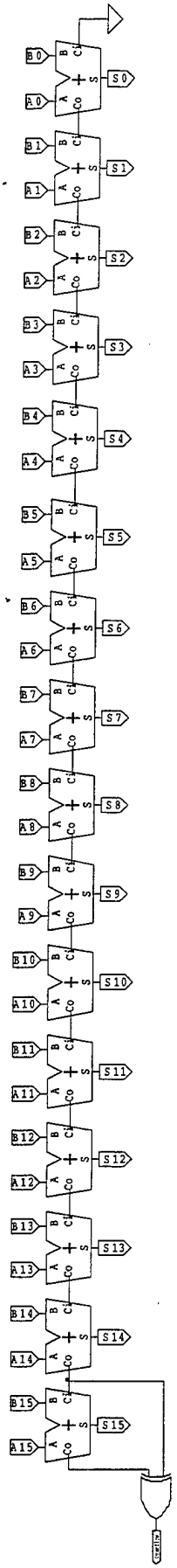
5-bit Adder


A4
A3
A2
A1
A0
B4
B3
B2
B1
B0
5-Bit + Adder
S4
S3
S2
S1
S0

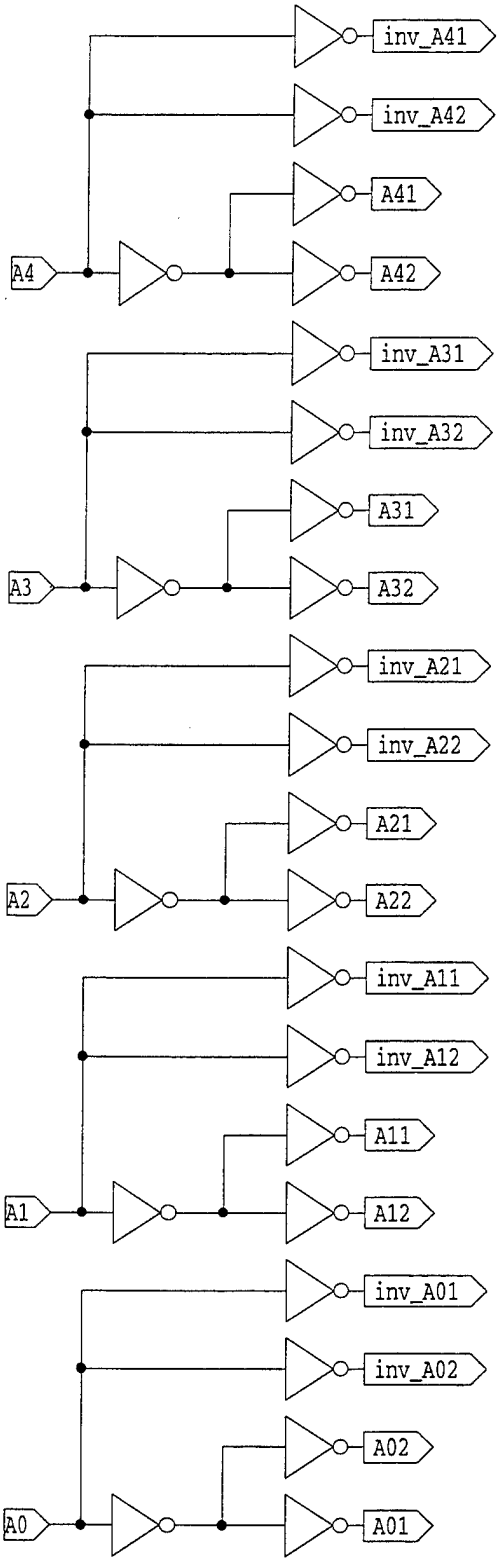

B0
A0
S0
B1
A1
S1
B2
A2
S2
B3
A3
S3
B4
A4
S4
Ci
Co
A
B
S

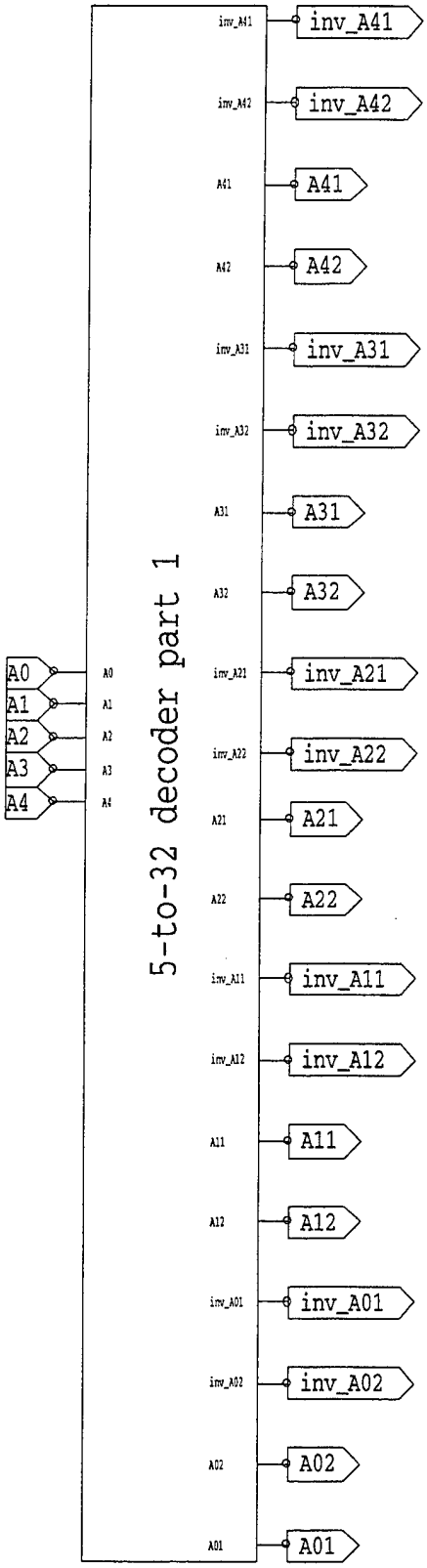
16-bit Adder


B0
B1
B2
B3
B4
B5
B6
B7
B8
B9
B10
B11
B12
B13
B14
B15
A0
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
A11
A12
A13
A14
A15
16 Bit Adder
Overflow
S0
S1
S2
S3
S4
S5
S6
S7
S8
S9
S10
S11
S12
S13
S14
S15
Overflow

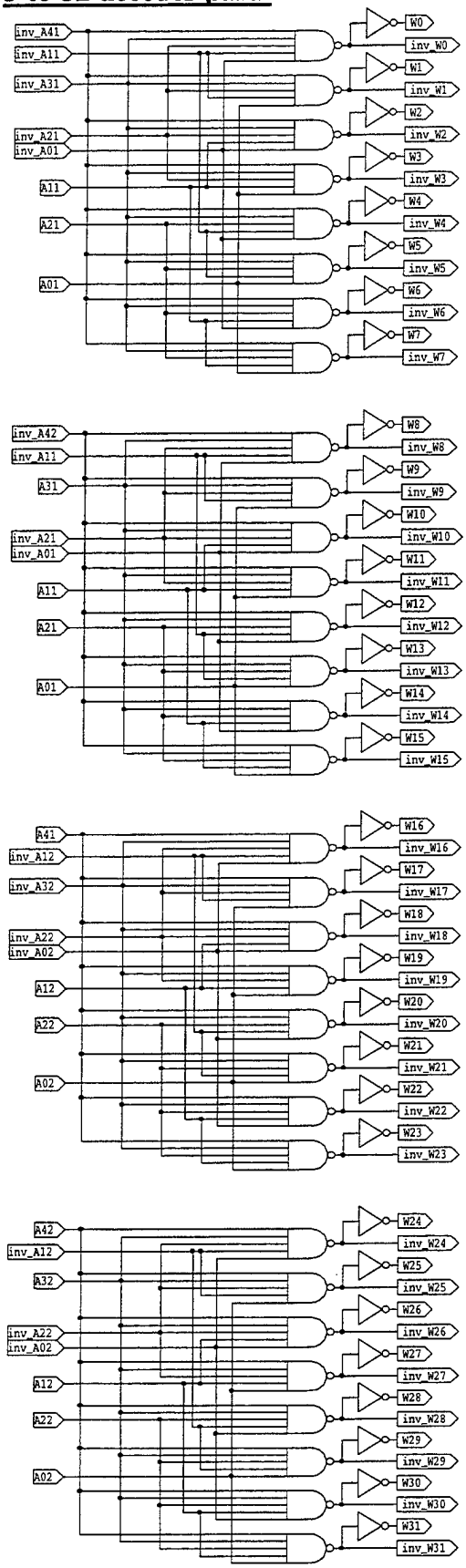


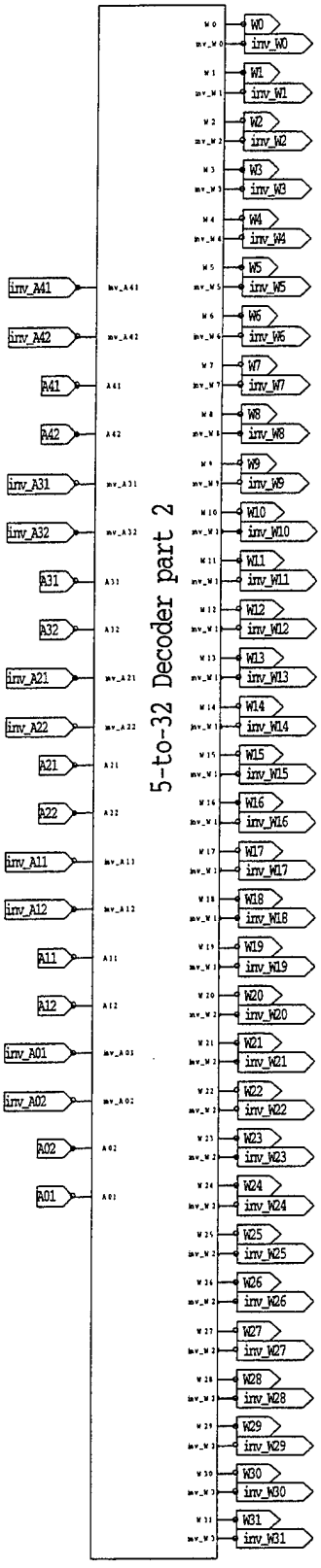
5-to-32 decoder part1



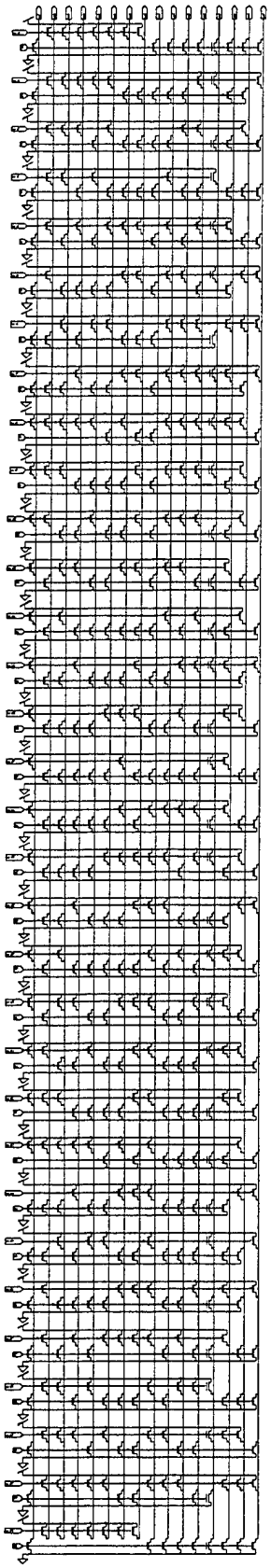


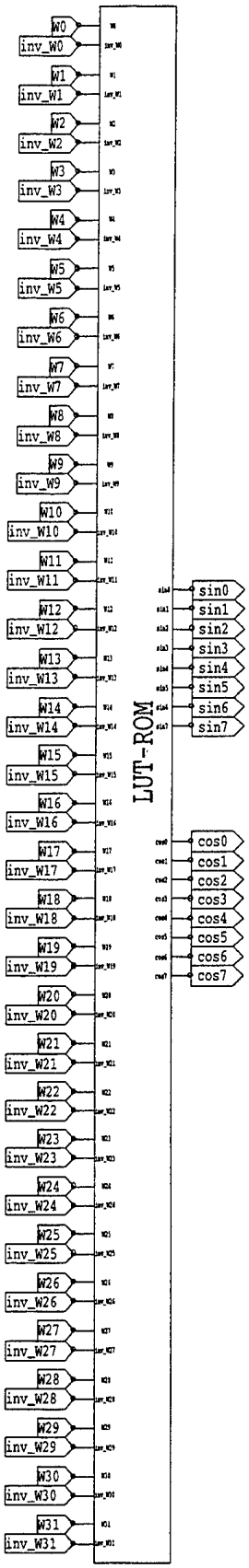
5-to-32 decoder part2



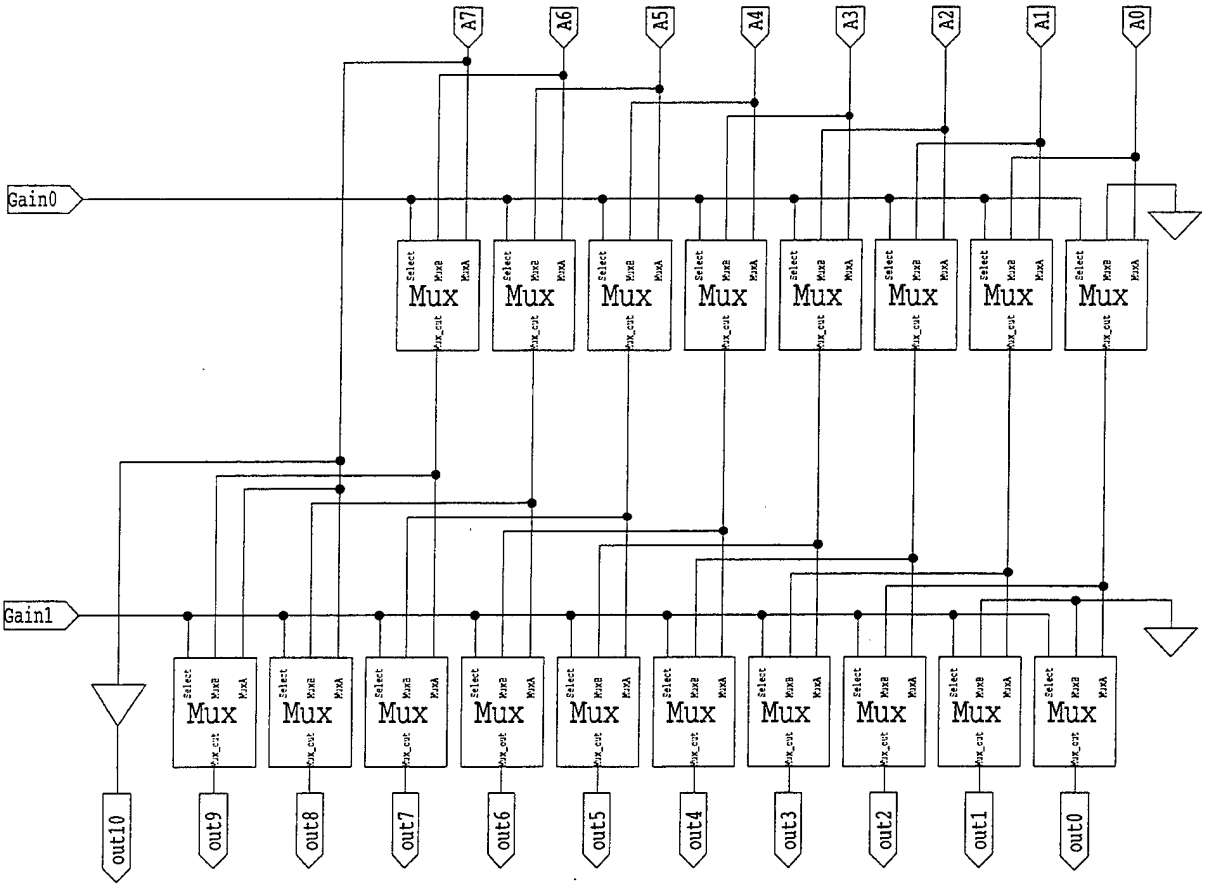


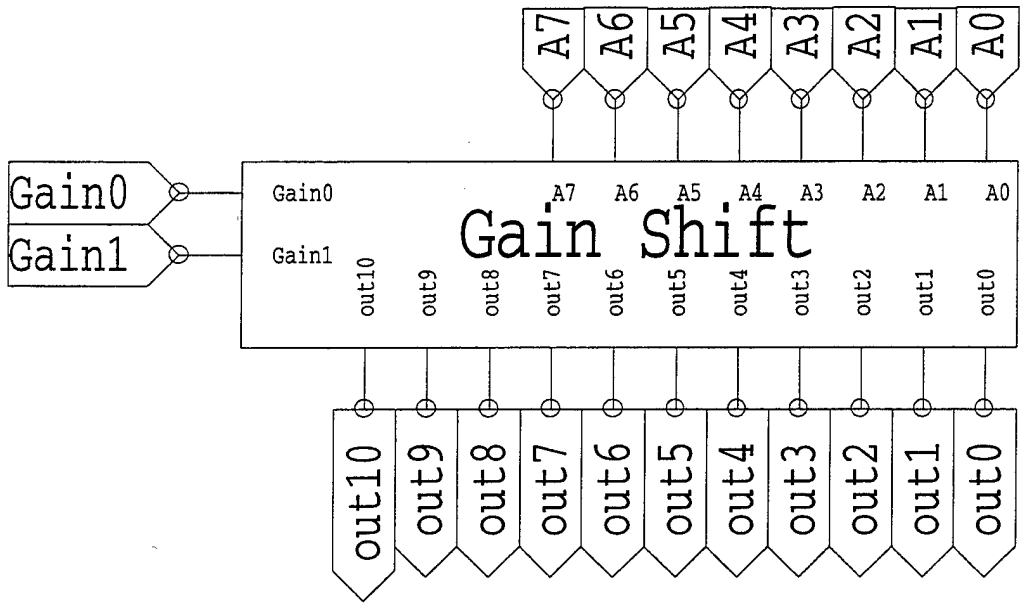
LUT ROM



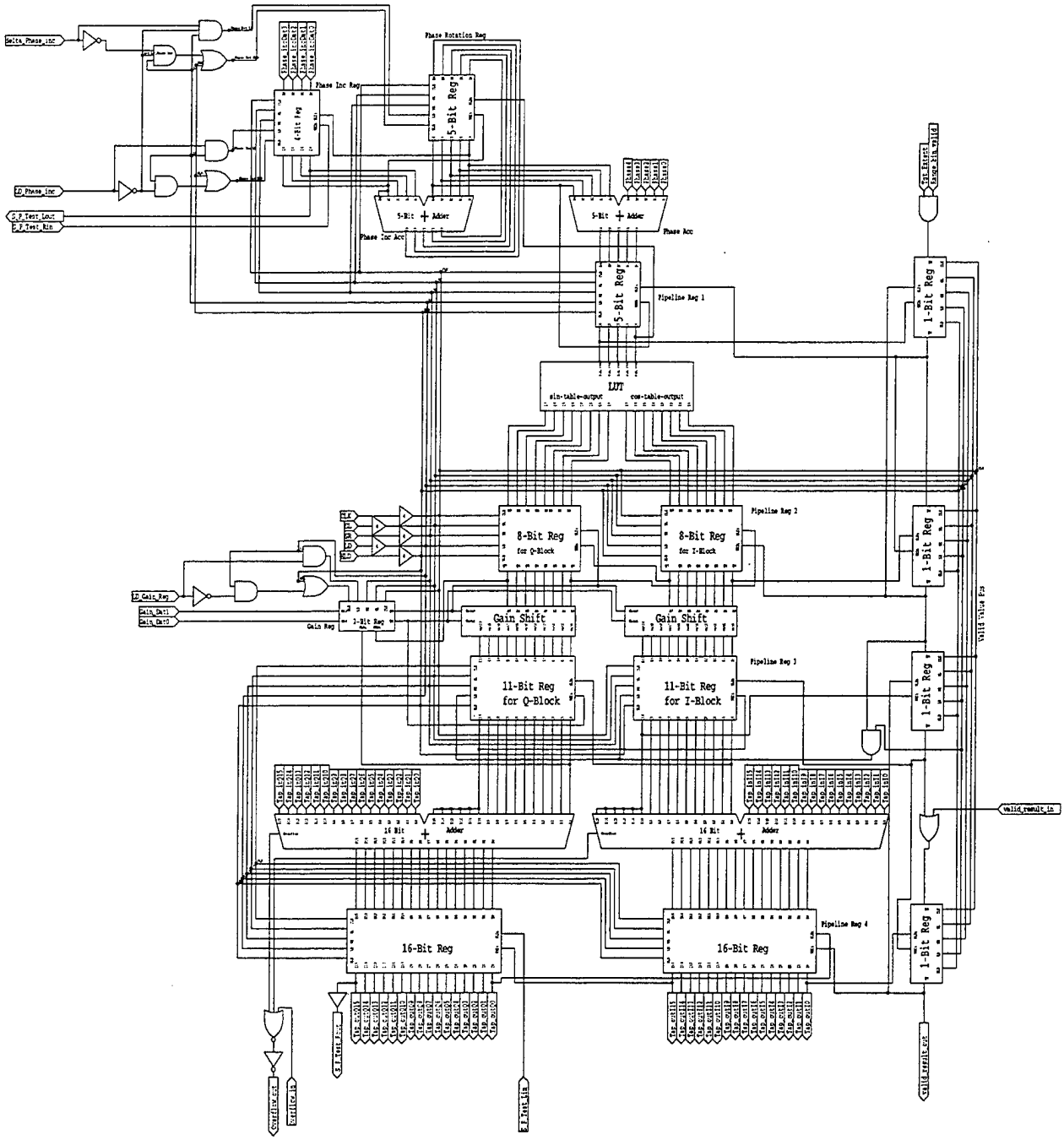

W0
inv_W0
W1
inv_W1
W2
inv_W2
W3
inv_W3
W4
inv_W4
W5
inv_W5
W6
inv_W6
W7
inv_W7
W8
inv_W8
W9
inv_W9
W10
inv_W10
W11
inv_W11
W12
inv_W12
W13
inv_W13
W14
inv_W14
W15
inv_W15
W16
inv_W16
W17
inv_W17
W18
inv_W18
W19
inv_W19
W20
inv_W20
W21
inv_W21
W22
inv_W22
W23
inv_W23
W24
inv_W24
W25
inv_W25
W26
inv_W26
W27
inv_W27
W28
inv_W28
W29
inv_W29
W30
inv_W30
W31
inv_W31
LUT-ROM
sin0
sin1
sin2
sin3
sin4
sin5
sin6
sin7
cos0
cos1
cos2
cos3
cos4
cos5
cos6
cos7

Gain Shift

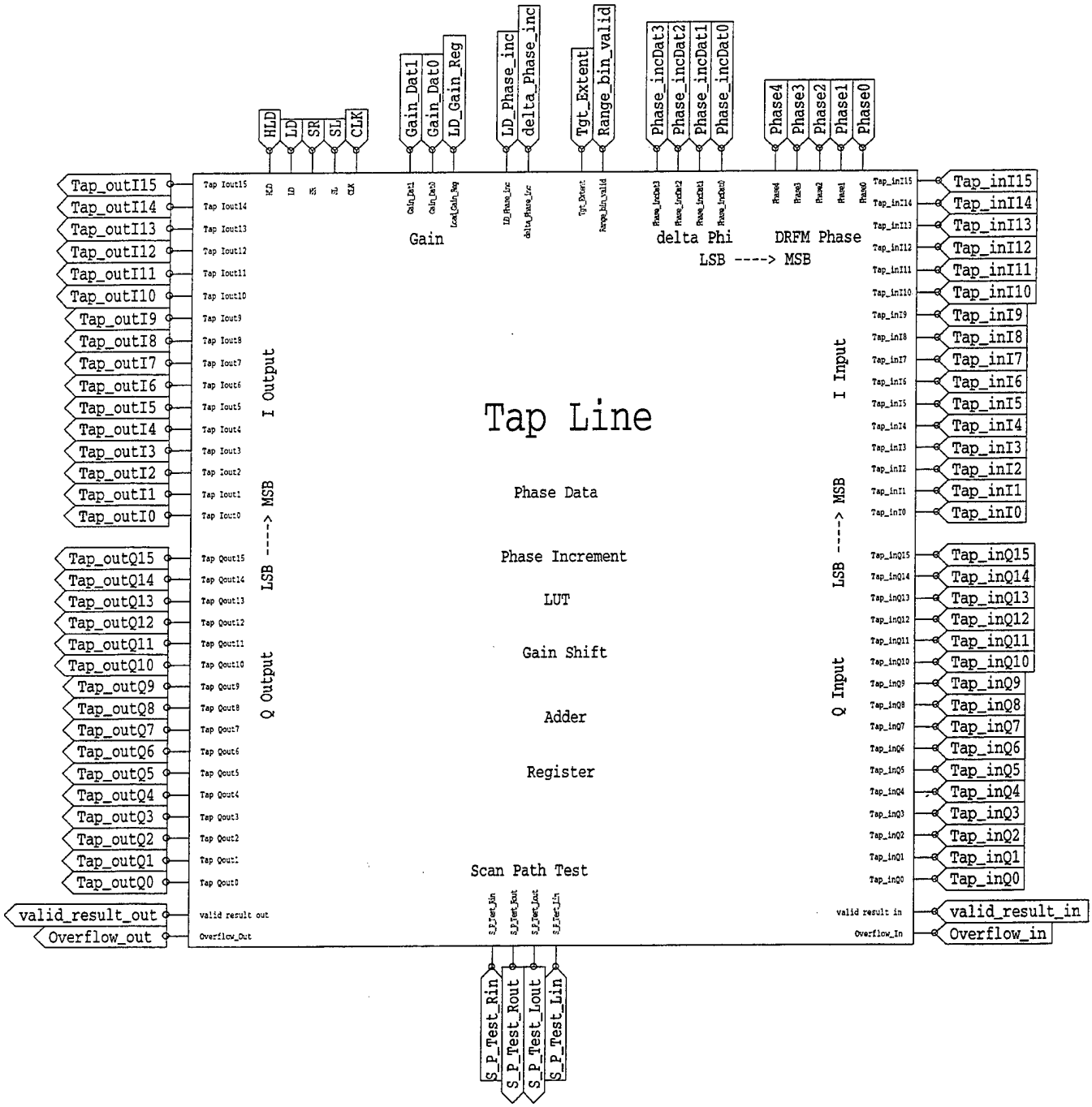

A7
A6
A5
A4
A3
A2
A1
A0
Gain0
Gain1
Select
MuxB
MuxA
Mux
Mux_out
out10
out9
out8
out7
out6
out5
out4
out3
out2
out1
out0


A7
A6
A5
A4
A3
A2
A1
A0
Gain0
Gain1
Gain Shift
out10
out9
out8
out7
out6
out5
out4
out3
out2
out1
out0

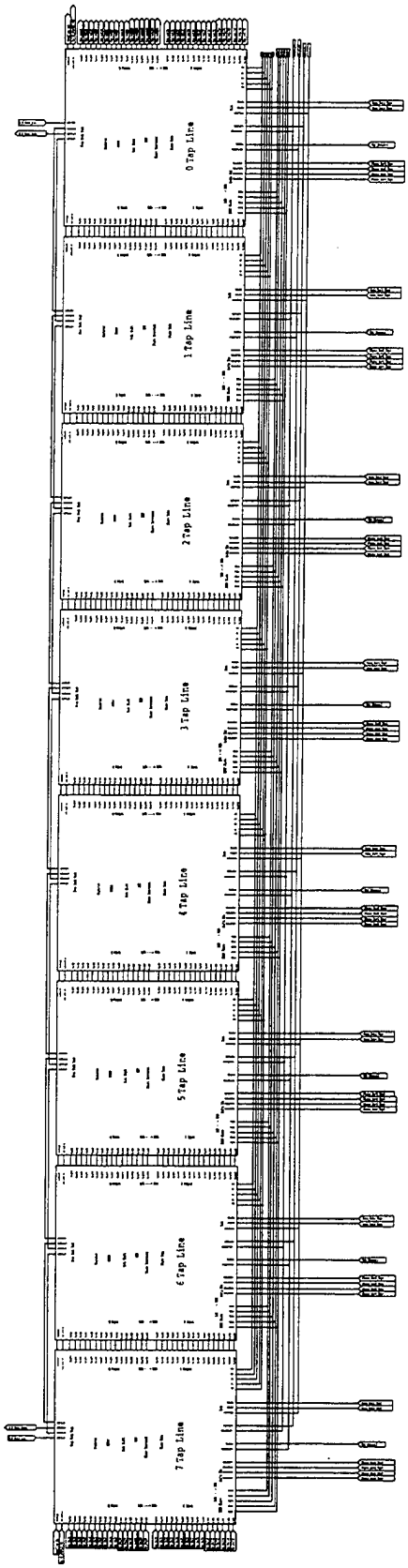
Level 3
Tapline



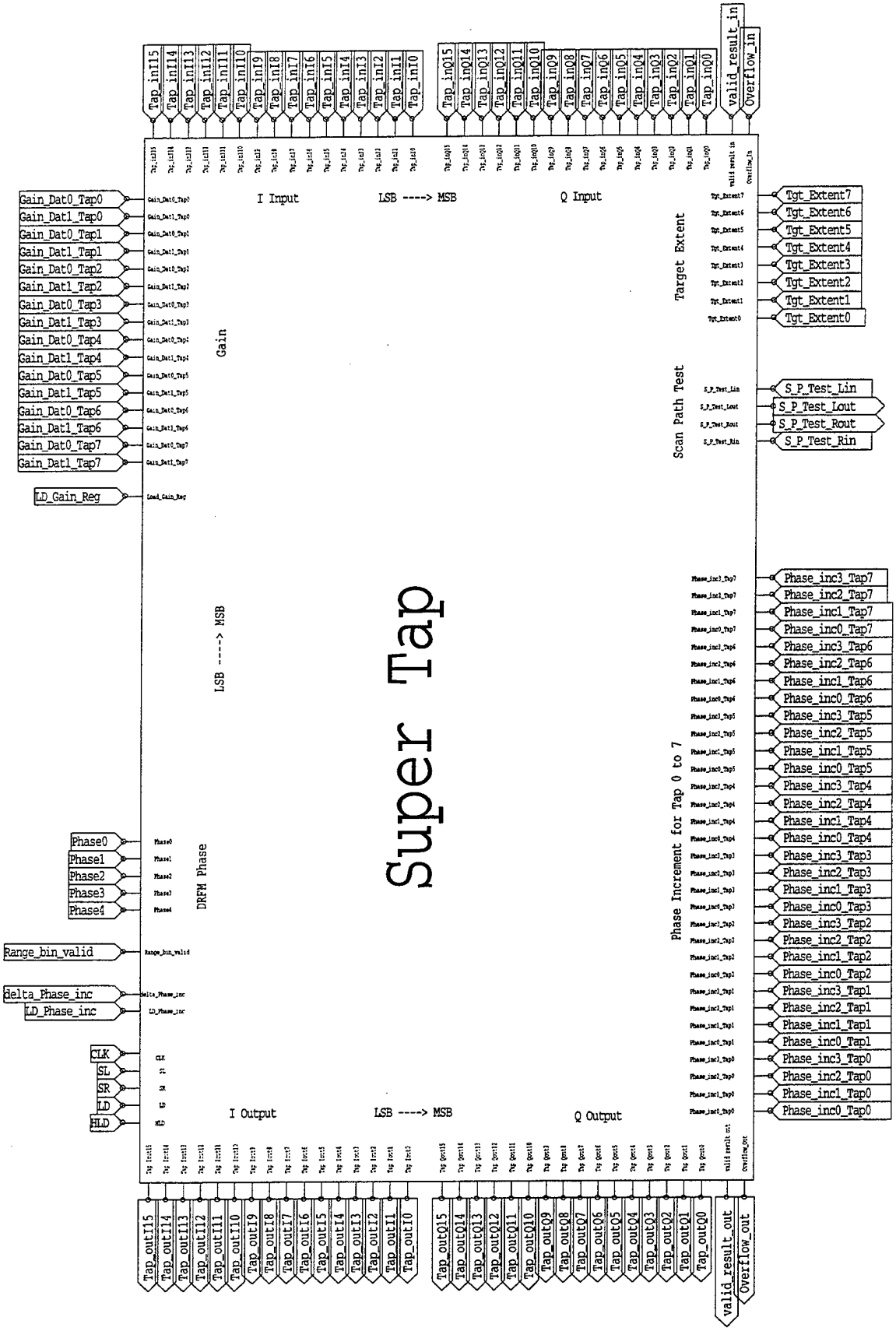
Tapline symbol



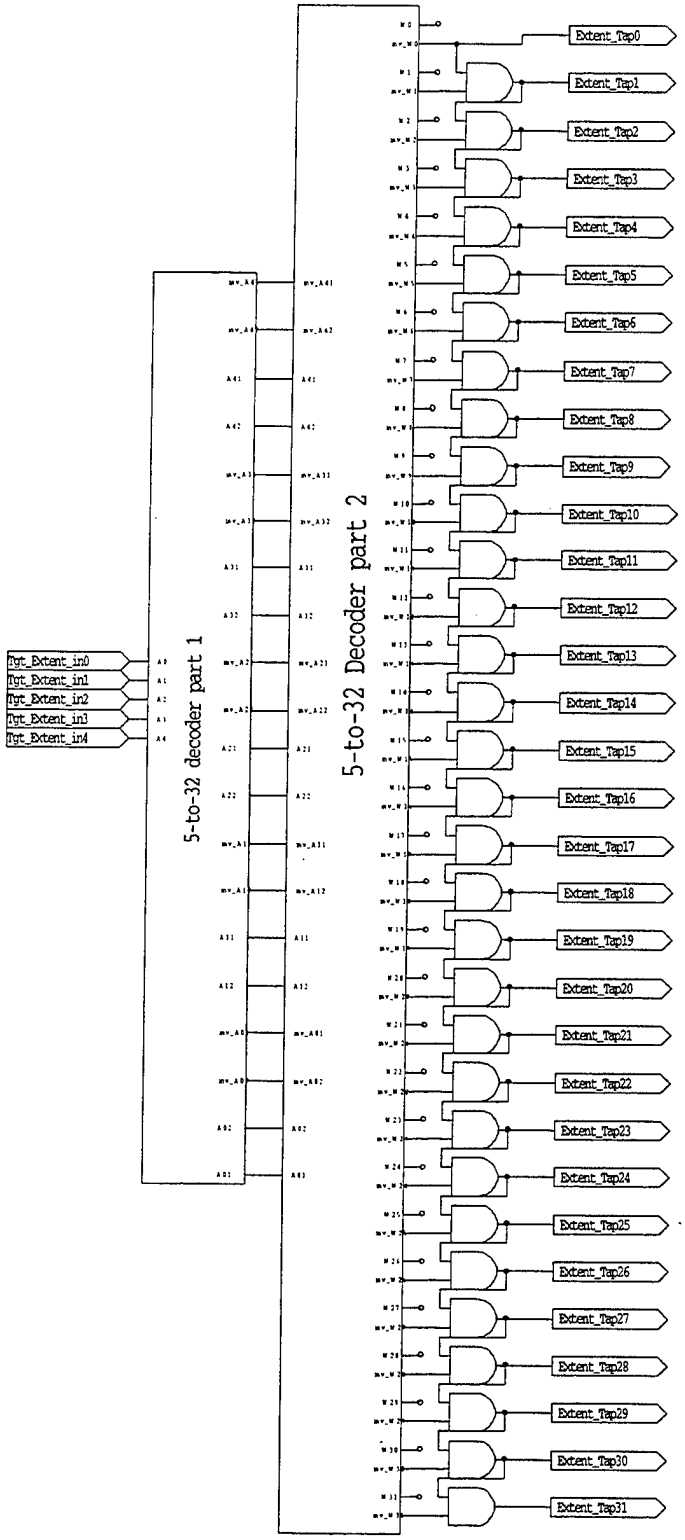
Level 4
Supertap



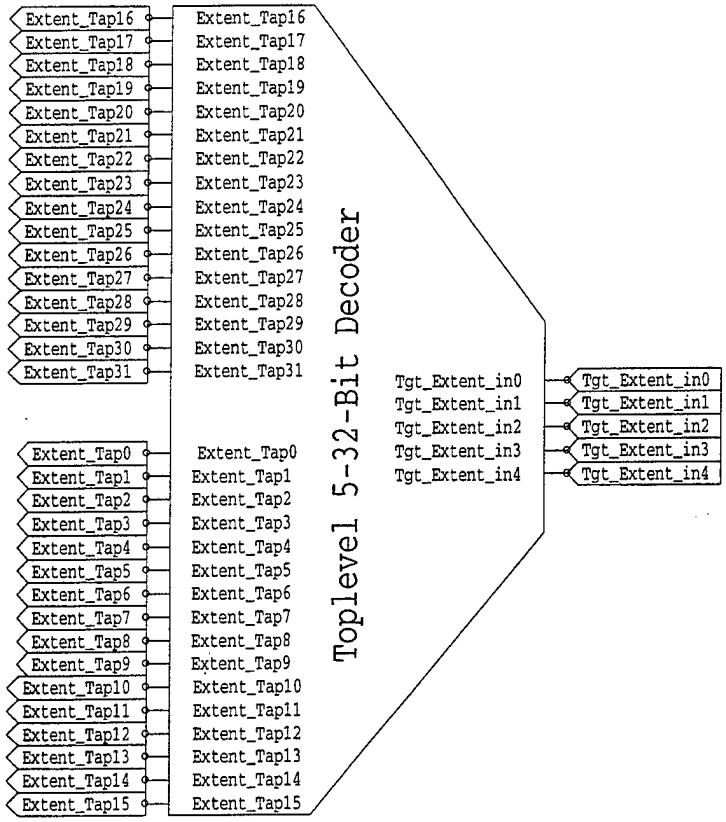
Supertap symbol



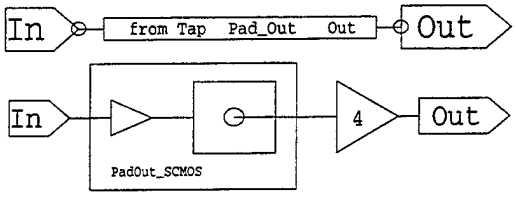
Level 5
Toplevel 5-to-32 Decoder



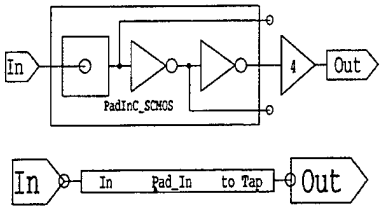
Toplevel 5-to-32 Decoder symbol



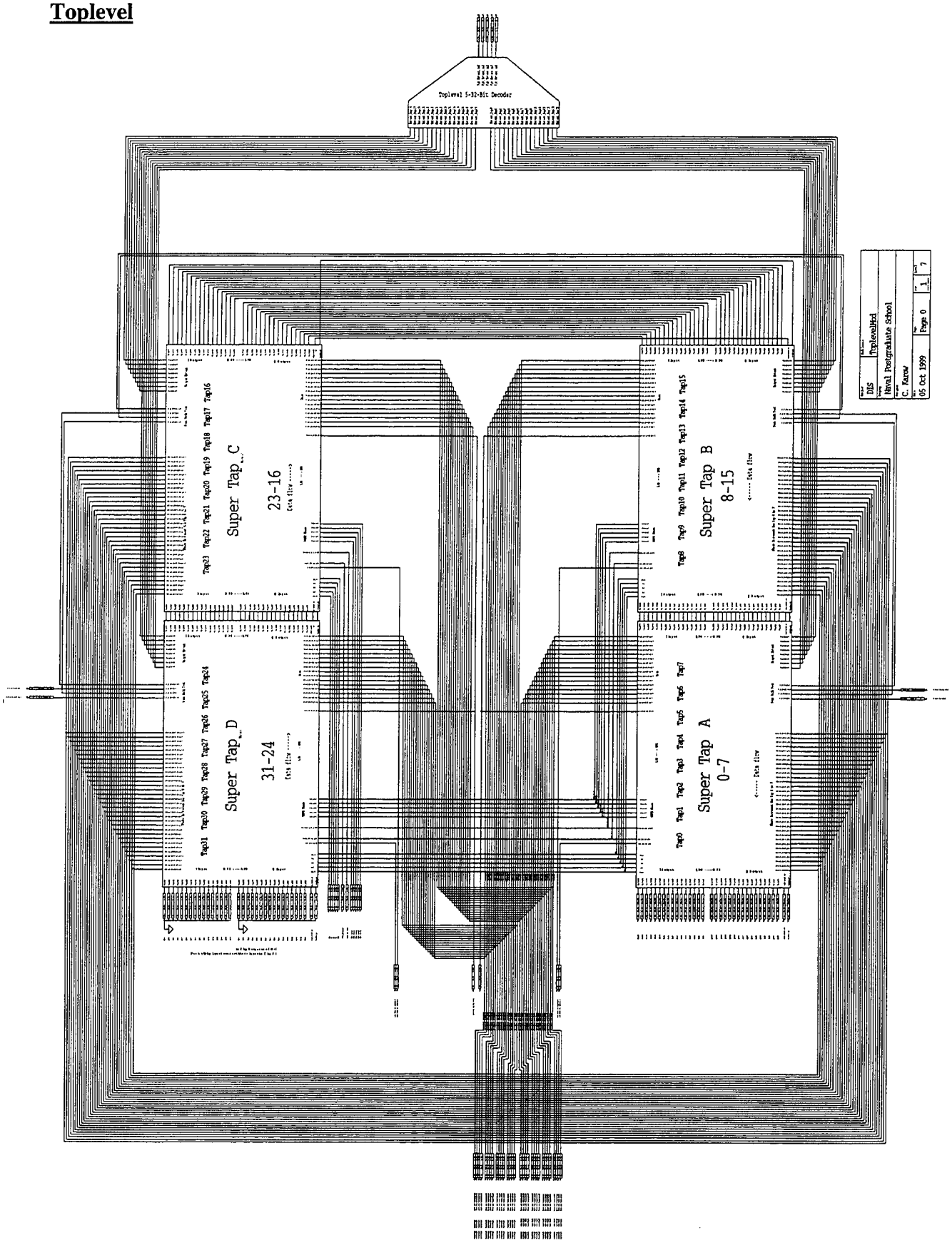
Input Pad



Output Pad



Toplevel


Toplevel 5-32-Bit Decoder
Super Tap C
23-16
Super Tap B
8-15
Super Tap D
31-24
Super Tap A
0-7
DIS
TopLevelMod
Naval Postgraduate School
C. Karow
05 Oct 1999
Page 0

Appendix D – T-Spice Simulation Files and Hardlimiter m-file

This Appendix includes all files and test vectors used for the 2-bit register-test and the2-Tapline-test. Furthermore the hard limiter script file for Matlab is attached.

2-bit Register-test:

Test Vectors:

VinD0 D0 Gnd bit ({0100101110100100000000000000001111} on=5.0 off=0.0 pw=80n rt=0.1n ft=0.1n
VinD1 D1 Gnd bit ({00000000000000001001011101001011111} on=5.0 off=0.0 pw=80n rt=0.1n ft=0.1n
VinSRDi SRDi Gnd bit ({10110110110110110110110110110100 11} on=5.0 off=0.0 pw=80n rt=0.1n ft=0.1n
VinSLDi SLDi Gnd bit ({11011011011011011011011011011000 11} on=5.0 off=0.0 pw=80n rt=0.1n ft=0.1n
VinHLD HLD Gnd bit ({00000011100000000000011100000010 00} on=5.0 off=0.0 pw=80n rt=0.1n ft=0.1n
VinLD LD Gnd bit ({00011100000000000011100000000100} on=5.0 off=0.0 pw=80n rt=0.1n ft=0.1n
VinSR SR Gnd bit ({00000000011100000000000011100000 10} on=5.0 off=0.0 pw=80n rt=0.1n ft=0.1n
VinSL SL Gnd bit ({00000000000011100000000000011100 01} on=5.0 off=0.0 pw=80n rt=0.1n ft=0.1n
VinCLK CLK Gnd bit ({01} on=5.0 off=0.0 pw=40n rt=0.1n ft=0.1n

Inputs and Outputs

```
D 0         0 1 0 0 1 0 1 1 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1
D 1         0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 1 1 0 1 0 0 1 0 1 1 1 1
S R D i     1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 0 0 1 1
S L D i     1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 0 1 1
H L D       0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 1 0 0 0
L D         0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 1 0 0
S R         0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 1 0
S L         0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 1
C L K       0 1

Q 0         0 0 0 0 1 0 0 0 0 1 0 1 0 1 1 0 0 0 0 0 0 0 0 0 1 0 1 0 1 1 1 1 1 1
Q 1         0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 1 0 0 0 0 0 1 0 1 1 0 0 1 1 1
```

Spice file for T-Spice simulation

```
* SPICE netlist written by S-Edit Win32 6.00
* 2-Bit-Register
* Written on Oct 20, 1999 at 11:07:37
Vdd Vdd Gnd DC 5
.include "D:\Chris\Thesis\schematics\testfiles\Register\2Bit\input_table2Reg.md"
.options prtdel=80n
.tran 10n 800n start=70n
.print tran "D:\Chris\Thesis\Schematics\testfiles\Register\2Bit\Inputs.out" V(CLK) V(SL) V(SR)
            V(SLDi) V(SRDi) V(LD) V(HLD) V(D0) V(D1)
.print tran "D:\Chris\Thesis\Schematics\testfiles\Register\2Bit\Outputs.out" V(Q0) V(Q1)
```

```
.param l=0.05u
.include "D:\Chris\Thesis\ModelParammod.md"
*
* Waveform probing commands
.probe
.options probefilename="D:\Chris\Thesis\Schematics\Testfiles\DISsept11C.dat"
+ probesdbfile="D:\Chris\Thesis\Schematics\DISsept11C.sdb"
+ probetopmodule="2-Bit Register"

* No Ports in cell: Legend
* End of module with no ports: Legend
* No Ports in cell: Legend_add
* End of module with no ports: Legend_add

.SUBCKT Buf4 A Out Gnd Vdd
M2 1 A Gnd Gnd NMOS W='18*l' L='7*l' AS='55.5652*l*l' AD='108*l*l' PS='26.6087*l'
PD='48*l' M=1
M4 Out 1 Gnd Gnd NMOS W='14*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=4
M1 1 A Vdd Vdd PMOS W='18*l' L='7*l' AS='55.5652*l*l' AD='108*l*l' PS='26.6087*l'
PD='48*l' M=1
M3 Out 1 Vdd Vdd PMOS W='28*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=4
.ENDS

* No Ports in cell: PageID_Tanner
* End of module with no ports: PageID_Tanner

.SUBCKT DFF Clk Data Q QB Gnd Vdd
M21 10 CB QB Gnd NMOS W='20*l' L='7*l' AS='60*l*l' AD='78.3333*l*l' PS='26*l'
PD='29.1667*l' M=1
M22 QB Q Gnd Gnd NMOS W='28*l' L='7*l' AS='84*l*l' AD='109.667*l*l' PS='34*l'
PD='40.8333*l' M=1
M17 10 C 11 Gnd NMOS W='20*l' L='7*l' AS='60*l*l' AD='20*l*l' PS='26*l' PD='22*l' M=1
M18 11 6 Gnd Gnd NMOS W='20*l' L='7*l' AS='20*l*l' AD='84*l*l' PS='22*l' PD='54*l' M=1
M11 4 C 8 Gnd NMOS W='19*l' L='7*l' AS='57*l*l' AD='57*l*l' PS='25*l' PD='25*l' M=1
M12 8 6 Gnd Gnd NMOS W='19*l' L='7*l' AS='57*l*l' AD='57*l*l' PS='25*l' PD='25*l' M=1
M7 4 CB 5 Gnd NMOS W='19*l' L='7*l' AS='57*l*l' AD='19*l*l' PS='25*l' PD='21*l' M=1
M8 5 Data Gnd Gnd NMOS W='19*l' L='7*l' AS='19*l*l' AD='126.871*l*l' PS='21*l'
PD='51.4839*l' M=1
M14 6 4 Gnd Gnd NMOS W='19*l' L='7*l' AS='84*l*l' AD='57*l*l' PS='50*l' PD='25*l' M=1
M24 Q 10 Gnd Gnd NMOS W='28*l' L='7*l' AS='148*l*l' AD='84*l*l' PS='68*l' PD='34*l'
M=1
M2 CB Clk Gnd Gnd NMOS W='6*l' L='7*l' AS='40.0645*l*l' AD='36*l*l' PS='16.2581*l'
PD='24*l' M=1
M4 C CB Gnd Gnd NMOS W='6*l' L='7*l' AS='40.0645*l*l' AD='36*l*l' PS='16.2581*l'
PD='24*l' M=1
* Page Size:  5x7
* S-Edit  D Flip Flop
* Designed by: D.Gunawan, J.Luo  Aug 26, 1999  21:32:22
* Schematic generated by S-Edit
```

```
* from file D:\Chris\Thesis\Schematics\DISsept11C / module DFF / page Page0
M19 QB Q Vdd Vdd PMOS W='28*l' L='7*l' AS='84*l*l' AD='109.667*l*l' PS='34*l'
PD='40.8333*l' M=1
M20 10 C QB Vdd PMOS W='20*l' L='7*l' AS='78.3333*l*l' AD='60*l*l' PS='29.1667*l'
PD='26*l' M=1
M15 9 6 Vdd Vdd PMOS W='20*l' L='7*l' AS='20*l*l' AD='81*l*l' PS='22*l' PD='52*l' M=1
M16 10 CB 9 Vdd PMOS W='20*l' L='7*l' AS='60*l*l' AD='20*l*l' PS='26*l' PD='22*l' M=1
M9 7 6 Vdd Vdd PMOS W='19*l' L='7*l' AS='57*l*l' AD='19*l*l' PS='25.6757*l' PD='21*l'
M=1
M10 4 CB 7 Vdd PMOS W='19*l' L='7*l' AS='19*l*l' AD='94*l*l' PS='21*l' PD='33*l' M=1
M5 3 Data Vdd Vdd PMOS W='19*l' L='7*l' AS='19*l*l' AD='158.742*l*l' PS='21*l'
PD='51.4839*l' M=1
M6 4 C 3 Vdd PMOS W='19*l' L='7*l' AS='94*l*l' AD='19*l*l' PS='33*l' PD='21*l' M=1
M13 6 4 Vdd Vdd PMOS W='18*l' L='7*l' AS='96*l*l' AD='54*l*l' PS='50*l' PD='24.3243*l'
M=1
M23 Q 10 Vdd Vdd PMOS W='28*l' L='7*l' AS='148*l*l' AD='84*l*l' PS='68*l' PD='34*l' M=1
M1 CB Clk Vdd Vdd PMOS W='6*l' L='7*l' AS='50.129*l*l' AD='36*l*l' PS='16.2581*l'
PD='24*l' M=1
M3 C CB Vdd Vdd PMOS W='6*l' L='7*l' AS='36*l*l' AD='50.129*l*l' PS='24*l'
PD='16.2581*l' M=1
.ENDS

.SUBCKT NAND4 A B C D Out Gnd Vdd
M5 Out D 1 Gnd NMOS W='14*l' L='7*l' AS='148*l*l' AD='84*l*l' PS='68*l' PD='34*l' M=1
M6 1 C 2 Gnd NMOS W='14*l' L='7*l' AS='84*l*l' AD='84*l*l' PS='34*l' PD='34*l' M=1
M7 2 B 3 Gnd NMOS W='14*l' L='7*l' AS='84*l*l' AD='84*l*l' PS='34*l' PD='34*l' M=1
M8 3 A Gnd Gnd NMOS W='14*l' L='7*l' AS='84*l*l' AD='144*l*l' PS='34*l' PD='68*l' M=1
* Page Size: 5x7
* S-Edit 4-Input NAND Gate (TIB)
* Designed by: J. Luo Oct 19, 1999 12:51:14
* Schematic generated by S-Edit
* from file D:\Chris\Thesis\Schematics\DISsept11C / module NAND4 / page Page0
M1 Out A Vdd Vdd PMOS W='28*l' L='7*l' AS='84*l*l' AD='144*l*l' PS='34*l' PD='68*l' M=1
M2 Out B Vdd Vdd PMOS W='28*l' L='7*l' AS='84*l*l' AD='84*l*l' PS='34*l' PD='34*l' M=1
M3 Out C Vdd Vdd PMOS W='28*l' L='7*l' AS='84*l*l' AD='84*l*l' PS='34*l' PD='34*l' M=1
M4 Out D Vdd Vdd PMOS W='28*l' L='7*l' AS='144*l*l' AD='84*l*l' PS='68*l' PD='34*l' M=1
.ENDS

.SUBCKT NAND2 A B Out Gnd Vdd
M3 Out B 1 Gnd NMOS W='14*l' L='7*l' AS='148*l*l' AD='84*l*l' PS='68*l' PD='34*l' M=1
M4 1 A Gnd Gnd NMOS W='14*l' L='7*l' AS='84*l*l' AD='144*l*l' PS='34*l' PD='68*l' M=1
* Page Size: 5x7
* S-Edit 2-Input NAND Gate (TIB)
* Designed by: J. Luo Oct 6, 1999 18:59:26
* Schematic generated by S-Edit
* from file D:\Chris\Thesis\Schematics\DISsept11C / module NAND2 / page Page0
M2 Out B Vdd Vdd PMOS W='28*l' L='7*l' AS='144*l*l' AD='84*l*l' PS='68*l' PD='34*l' M=1
M1 Out A Vdd Vdd PMOS W='28*l' L='7*l' AS='84*l*l' AD='144*l*l' PS='34*l' PD='68*l' M=1
.ENDS
```

```
.SUBCKT Register_Cell CLK Di HLD LD Q QB SL SLDi SR SRDi Gnd Vdd
XDFF_1 CLK N3 Q QB Gnd Vdd DFF
XNAND2_1 Q HLD N8 Gnd Vdd NAND2
XNAND2_2 SRDi SR N7 Gnd Vdd NAND2
XNAND2_3 SLDi SL N6 Gnd Vdd NAND2
XNAND2_4 Di LD N5 Gnd Vdd NAND2
XNAND4_1 N8 N7 N6 N5 N3 Gnd Vdd NAND4
.ENDS

* Main circuit: 2-Bit Register
XBuf4_1 LD N6 Gnd Vdd Buf4
XBuf4_2 HLD N7 Gnd Vdd Buf4
XBuf4_3 SR N4 Gnd Vdd Buf4
XBuf4_4 SL N3 Gnd Vdd Buf4
XBuf4_5 CLK N5 Gnd Vdd Buf4
XRegister_Cell_1 N5 D0 N7 N6 Q0 N1 N3 Q1 N4 SRDi Gnd Vdd Register_Cell
XRegister_Cell_2 N5 D1 N7 N6 Q1 N2 N3 SLDi N4 Q0 Gnd Vdd Register_Cell
* End of main circuit: 2-Bit Register
```

2-Tapline-test:

Test vector:

Control Signals

VinCLK CLK Gnd bit ({01} on=5.0 off=0.0 pw=200n rt=0.1n ft=0.1n

VinHLD HLD Gnd bit ({0} on=5.0 off=0.0 pw=400n rt=0.1n ft=0.1n

VinLD LD Gnd bit ({0 111111111111111 1 111111111111111 1 111111111111111 1 111111111111111 1 111111111111111 1 111111111111111 1 111111111111111 1 111111111111111 1 111111111111111 1111111} on=5.0 off=0.0 pw=400n rt=0.1n ft=0.1n

VinSR SR Gnd bit ({0} on=5.0 off=0.0 pw=400n rt=0.1n ft=0.1n

VinSL SL Gnd bit ({0} on=5.0 off=0.0 pw=400n rt=0.1n ft=0.1n

VinTgt_Extent0 Tgt_Extent0 Gnd bit ({1} on=5.0 off=0.0 pw=400n rt=0.1n ft=0.1n

VinTgt_Extent1 Tgt_Extent1 Gnd bit ({1} on=5.0 off=0.0 pw=400n rt=0.1n ft=0.1n

VinRange_bin_valid Range_bin_valid Gnd bit ({0 111111111111111 0 111111111111111 0 111111111111111 0 111111111111111 0 111111111111111 0 111111111111111 0 111111111111111 0 111111111111111 0 111111111111111 0 111111111111111 0 111111111111111 0000000} on=5.0 off=0.0 pw=400n rt=0.1n ft=0.1n

Vinvalid_result_in valid_result_in Gnd bit ({0} on=5.0 off=0.0 pw=400n rt=0.1n ft=0.1n

VinOverflow_in Overflow_in Gnd bit ({0} on=5.0 off=0.0 pw=400n rt=0.1n ft=0.1n

Scan Path Test

VinS_P_Test_Rin S_P_Test_Rin Gnd bit ({0} on=5.0 off=0.0 pw=400n rt=0.1n ft=0.1n

VinS_P_Test_Lin S_P_Test_Lin Gnd bit ({0} on=5.0 off=0.0 pw=400n rt=0.1n ft=0.1n

DRFM Phase Data Inputs

VinPhase4 Phase4 Gnd bit ({0 000001110100010 0 000001110100010 0 000001110100010 0 000001110100010 0 000001110100010 0 000001110100010 0 000001110100010 0 000001110100010 0 000001110100010 0 000001110100010 0 000001110100010 0000000} on=5.0 off=0.0 pw=400n rt=0.1n ft=0.1n

VinPhase3 Phase3 Gnd bit ({0 000101010011111 0 000101010011111 0 000101010011111 0 000101010011111 0 000101010011111 0 000101010011111 0 000101010011111 0 000101010011111 0 000101010011111 0 000101010011111 0 000101010011111 0000000} on=5.0 off=0.0 pw=400n rt=0.1n ft=0.1n

VinPhase2 Phase2 Gnd bit ({0 001110100100110 0 001110100100110 0 001110100100110 0 001110100100110 0 001110100100110 0 001110100100110 0 001110100100110 0 001110100100110 0 001110100100110 0 001110100100110 0 001110100100110 0000000} on=5.0 off=0.0 pw=400n rt=0.1n ft=0.1n

VinPhase1 Phase1 Gnd bit ({0 000100000011111 0 000100000011111 0 000100000011111 0 000100000011111 0 000100000011111 0 000100000011111 0 000100000011111 0 000100000011111 0 000100000011111 0 000100000011111 0 000100000011111 0000000} on=5.0 off=0.0 pw=400n rt=0.1n ft=0.1n

VinPhase0 Phase0 Gnd bit ({0 001101010000110 0 001101010000110 0 001101010000110 0 001101010000110 0 001101010000110 0 001101010000110 0 001101010000110 0 001101010000110 0 001101010000110 0 001101010000110 0 001101010000110 0000000} on=5.0 off=0.0 pw=400n rt=0.1n ft=0.1n

Phase Increment Data and Controls

Vindelta_Phase_inc delta_Phase_inc Gnd bit ({0 000000000000000 1 000000000000000 1 000000000000000 1 000000000000000 1 000000000000000 1 000000000000000 1 000000000000000 1 000000000000000 1 000000000000000 1 000000000000000 0000000} on=5.0 off=0.0 pw=400n rt=0.1n ft=0.1n

VinLD_Phase_inc LD_Phase_inc Gnd bit ({0 100000000000000 0 000000000000000 0 000000000000000 0 000000000000000 0 000000000000000 0 000000000000000 0 000000000000000 0 000000000000000 0 000000000000000 0 000000000000000 0000000} on=5.0 off=0.0 pw=400n rt=0.1n ft=0.1n

VinPhase_inc3_Tap0 Phase_inc3_Tap0 Gnd bit ({0} on=5.0 off=0.0 pw=400n rt=0.1n ft=0.1n
VinPhase_inc2_Tap0 Phase_inc2_Tap0 Gnd bit ({0} on=5.0 off=0.0 pw=400n rt=0.1n ft=0.1n
VinPhase_inc1_Tap0 Phase_inc1_Tap0 Gnd bit ({1} on=5.0 off=0.0 pw=400n rt=0.1n ft=0.1n
VinPhase_inc0_Tap0 Phase_inc0_Tap0 Gnd bit ({0} on=5.0 off=0.0 pw=400n rt=0.1n ft=0.1n
VinPhase_inc3_Tap1 Phase_inc3_Tap1 Gnd bit ({0} on=5.0 off=0.0 pw=400n rt=0.1n ft=0.1n
VinPhase_inc2_Tap1 Phase_inc2_Tap1 Gnd bit ({1} on=5.0 off=0.0 pw=400n rt=0.1n ft=0.1n
VinPhase_inc1_Tap1 Phase_inc1_Tap1 Gnd bit ({0} on=5.0 off=0.0 pw=400n rt=0.1n ft=0.1n
VinPhase_inc0_Tap1 Phase_inc0_Tap1 Gnd bit ({0} on=5.0 off=0.0 pw=400n rt=0.1n ft=0.1n

Gain Coefficients and Control

ViLD_Gain_Reg LD_Gain_Reg Gnd bit ({0 100000000000000 0 100000000000000 0 100000000000000 0 100000000000000 0 100000000000000 0 100000000000000 0 100000000000000 0 100000000000000 0 100000000000000 0 100000000000000 0000000} on=5.0 off=0.0 pw=400n rt=0.1n ft=0.1n
ViGain_Dat1_Tap0 Gain_Dat1_Tap0 Gnd bit ({0} on=5.0 off=0.0 pw=400n rt=0.1n ft=0.1n
ViGain_Dat0_Tap0 Gain_Dat0_Tap0 Gnd bit ({1} on=5.0 off=0.0 pw=400n rt=0.1n ft=0.1n
ViGain_Dat1_Tap1 Gain_Dat1_Tap1 Gnd bit ({1} on=5.0 off=0.0 pw=400n rt=0.1n ft=0.1n
ViGain_Dat0_Tap1 Gain_Dat0_Tap1 Gnd bit ({0} on=5.0 off=0.0 pw=400n rt=0.1n ft=0.1n

The inputs to the higher Tapline was set to zero.

Outputs

CLK	Valid Result	I values	Q values	Pulse#	Sample#
1	0	0000000000000000	0000000000000000		
2	0	0000000000000000	0000000000000000		
3	0	0000000000000000	0000000000000000		
4	0	0000000000000000	0000000000000000		
5	1	0000000011111110	0000000000000000	1	1
6	1	0000001011111010	0000000000000000	1	2
7	1	0000001010000010	0000000011010110	1	3
8	1	0000000000010000	0000000111000100	1	4
9	1	1111111010110110	0000000011101000	1	5
10	1	0000000110110100	0000000010000010	1	6
11	1	0000000000010110	1111110101011100	1	7
12	1	1111111100100100	1111110110000010	1	8
13	1	0000000110101110	1111111000100100	1	9
14	1	0000000101100010	1111111100111000	1	10
15	1	1111111001011110	1111111101010100	1	11

16	1	1111111010110110	0000001010101100	1	12
17	1	1111111000101000	0000000111100000	1	13
18	1	1111111100000110	0000000000110000	1	14
19	1	0000000110001110	0000000011100100	1	15
20	1	1111111100100100	0000000111001000	1	15
21	1	0000000011101000	0000000001100100	2	1
22	1	0000001001000100	0000000111010100	2	2
23	1	0000000110000010	0000001001101010	2	3
24	1	1111111010010010	0000000110011100	2	4
25	1	1111111011011000	1111111110100110	2	5
26	1	0000000010010110	0000000101000000	2	6
27	1	0000000110010010	1111111001000100	2	7
28	1	0000000011111010	1111110101010100	2	8
29	1	0000001010111000	1111111110011100	2	9
30	1	0000000100011110	0000000001111100	2	10
31	1	1111111110001100	1111111010110000	2	11
32	1	1111110101011100	0000000100111100	2	12
33	1	1111110100101010	0000000001001100	2	13
34	1	1111111101111000	1111111011101010	2	14
35	1	0000000011100000	0000000111000100	2	15
36	1	1111111000011100	0000000010011000	2	15
37	1	0000000010101110	0000000010111000	3	1
38	1	0000000010010110	0000001010110000	3	2
39	1	1111111110101010	0000001011101100	3	3
40	1	1111110110000100	0000000001010000	3	4
41	1	1111111111011100	1111111100000100	3	5
42	1	1111111101101000	0000000011100100	3	6
43	1	0000001000010110	1111111101101010	3	7
44	1	0000001010001000	1111111001111100	3	8
45	1	0000001010011110	0000000100011100	3	9
46	1	0000000000001110	0000000011111110	3	10
47	1	0000000010101110	1111111100101100	3	11
48	1	1111110101010010	1111111101011000	3	12
49	1	1111110110000100	1111111001101000	3	13
50	1	0000000010111000	1111111010011000	3	14
51	1	1111111101011010	0000001001000000	3	15
52	1	1111111001000100	1111111100001100	3	15
53	1	0000000001011000	0000000011101110	4	1
54	1	1111111011011000	0000001000110110	4	2
55	1	1111110111100110	0000001000010000	4	3
56	1	1111110110101110	1111111010000100	4	4
57	1	0000000011101110	1111111101110100	4	5
58	1	1111111100000110	1111111111010000	4	6
59	1	0000000110010010	0000000011010110	4	7
60	1	0000001011111010	0000000000000000	4	8
61	1	0000000101100100	0000001010011010	4	9
62	1	1111111100000110	0000000001110010	4	10
63	1	0000000100000000	1111111111101000	4	11
64	1	1111111010001000	1111111000000000	4	12

65	1	1111111100010110	1111110100110100	4	13
66	1	0000000111100010	1111111101100110	4	14
67	1	1111110111010000	0000000101111000	4	15
68	1	1111111110000100	1111111000011000	4	15
69	1	1111111111110100	0000000011111100	5	1
70	1	1111110111111100	0000000011001100	5	2
71	1	1111110100101010	0000000001001010	5	3
72	1	1111111100011000	1111110100111100	5	4
73	1	0000000100111100	0000000010100100	5	5
74	1	1111111111000100	1111111010100010	5	6
75	1	0000000001010100	0000000101011000	5	7
76	1	0000001001010100	0000000110100010	5	8
77	1	1111111101111000	0000001011100100	5	9
78	1	1111111011011000	1111111101000000	5	10
79	1	0000000001111110	0000000011110110	5	11
80	1	0000000000101110	1111110111011010	5	12
81	1	0000000100000000	1111110101011000	5	13
82	1	0000001000100010	0000000011111010	5	14
83	1	1111110100101010	1111111110110110	5	15
84	1	0000000100001100	1111111001010100	5	15
85	1	1111111110010010	0000000011100100	6	1
86	1	1111111001011110	1111111101010100	6	2
87	1	1111110111010000	1111111010001000	6	3
88	1	0000000100001000	1111110100110110	6	4
89	1	0000000010101110	0000000101101100	6	5
90	1	0000000100011100	1111111010011100	6	6
91	1	1111111100111100	0000000011000010	6	7
92	1	0000000010111000	0000001010001000	6	8
93	1	1111110111010110	0000000111011000	6	9
94	1	1111111111000100	1111111000111110	6	10
95	1	1111111101001110	0000000100110000	6	11
96	1	0000000101010110	1111111011010100	6	12
97	1	0000001001001000	1111111010101110	6	13
98	1	0000000101100010	0000001000010100	6	14
99	1	1111110111100110	1111110111110000	6	15
100	1	0000000111110000	1111111110011100	6	15
101	1	1111111101000000	0000000010100100	7	1
102	1	1111111110001100	1111111010110000	7	2
103	1	1111111101011010	1111110111000000	7	3
104	1	0000001001111110	1111111010000000	7	4
105	1	1111111101010000	0000000111011100	7	5
106	1	0000001000100110	1111111110110110	7	6
107	1	1111111100011010	1111111101101000	7	7
108	1	1111111100000110	0000001000011110	7	8
109	1	1111110101011100	0000000000001100	7	9
110	1	0000000101001010	1111111000001100	7	10
111	1	1111111001000010	0000000001010100	7	11
112	1	0000000110010010	1111111111010100	7	12
113	1	0000001001111110	0000000000010000	7	13

114	1	1111111110110010	0000001010111100	7	14
115	1	1111111110101010	1111110100010100	7	15
116	1	0000000111010000	0000000011001000	7	15
117	1	1111111100001110	0000000001001100	8	1
118	1	0000000010101110	1111111100101100	8	2
119	1	0000000011100000	1111111000111100	8	3
120	1	0000001011011000	0000000000000000	8	4
121	1	1111110111101110	0000000100010110	8	5
122	1	0000001000100010	0000000011111010	8	6
123	1	0000000000001100	1111111010011100	8	7
124	1	1111111000101110	0000000011001010	8	8
125	1	1111111000110010	1111111010000100	8	9
126	1	0000001010001000	1111111101000100	8	10
127	1	1111111000101110	1111111011010110	8	11
128	1	0000000011010110	0000000011100010	8	12
129	1	0000000110011000	0000000101111000	8	13
130	1	1111110111101110	0000001000001010	8	14
131	1	0000000110000010	1111110110010110	8	15
132	1	0000000010110000	0000000111011100	8	15
133	1	1111111100000100	1111111111101000	9	1
134	1	0000000100000000	1111111111101000	9	2
135	1	0000000110001110	1111111100011100	9	3
136	1	0000001000001010	0000000110101100	9	4
137	1	1111110101101110	1111111101101000	9	5
138	1	0000000100011110	0000001001100100	9	6
139	1	0000000101011110	1111111011011100	9	7
140	1	1111111010001110	1111111101100100	9	8
141	1	1111111110110100	1111111000001100	9	9
142	1	0000001010101010	0000000010111000	9	10
143	1	1111111101010010	1111110110011010	9	11
144	1	1111111110101010	0000000011110010	9	12
145	1	0000000000100010	0000000111001000	9	13
146	1	1111110100001100	0000000001001000	9	14
147	1	0000001010000010	1111111100101010	9	15
148	1	1111111100100100	0000000111001000	9	16
149	1	1111111100100010	1111111110000110	10	1
150	1	0000000001111110	0000000011110110	10	2
151	1	0000000101010000	0000000001110100	10	3
152	1	0000000001111100	0000001000011010	10	4
153	1	1111111001000010	1111110111000100	10	5
154	1	1111111101001110	0000001011000000	10	6
155	1	0000001000101000	0000000000100110	10	7
156	1	1111111110110010	1111111011010100	10	8
157	1	0000000011110010	1111111010111110	10	9
158	1	0000000110110100	0000001001011110	10	10
159	1	0000000100011100	1111110101111100	10	11
160	1	1111111011101100	0000000000001000	10	12
161	1	1111111100010100	0000000011001010	10	13
162	1	1111110110001110	1111111001101100	10	14

163	1	0000001001110000	0000000010010000	10	15
164	1	1111111000011100	0000000010011000	10	16
165	0	0000000000000000	0000000000000000		
166	0	0000000000000000	0000000000000000		
167	0	0000000000000000	0000000000000000		
168	0	0000000000000000	0000000000000000		
169	0	0000000000000000	0000000000000000		
170	0	0000000000000000	0000000000000000		
171	0	0000000000000000	0000000000000000		

Spice file for T-Spice simulation

```
* SPICE netlist written by S-Edit Win32 6.00
* Written on Nov 19, 1999 at 13:57:23
Vdd Vdd Gn  DC 5
.include "D:\C  is\Thesis\schematics\testfiles\Tapline\2Tap_Test\input_tableTapTest.md"
.options  abstol=50n reltol=0.001 numnt=6 prtdel=400n
.tran 300n 68000n start=390n
.print tran "D:\Chris\Thesis\Schematics\testfiles\Tapline\2Tap_Test\CTRLInputs2.out" V(CLK)
            V(SL) V(SR) V(LD) V(HLD) V(delta_Phase_inc) V(Phase0) V(Phase1) V(Phase2)
            V(Phase3) V(Phase4) V(LD_Phase_inc) V(Tgt_Extent0) V(Tgt_Extent1)
            V(Range_bin_valid) V(S_P_Test_Rin) V(S_P_Test_Lin)
.print tran "D:\Chris\Thesis\Schematics\testfiles\Tapline\2Tap_Test\Inputs2.out"
            V(Phase_inc0_Tap0) V(Phase_inc1_Tap0) V(Phase_inc2_Tap0) V(Phase_inc3_Tap0)
            V(Phase_inc0_Tap1) V(Phase_inc1_Tap1) V(Phase_inc2_Tap1) V(Phase_inc3_Tap1)
            V(LD_Gain_Reg) V(Gain_Dat0_Tap0) V(Gain_Dat1_Tap0) V(Gain_Dat0_Tap1)
            V(Gain_Dat1_Tap1)
.print tran "D:\Chris\Thesis\Schematics\testfiles\Tapline\2Tap_Test\16Adder1Inputs2.out"
            V(Tap_inQ0) V(Tap_inQ1) V(Tap_inQ2) V(Tap_inQ3) V(Tap_inQ4) V(Tap_inQ5)
            V(Tap_inQ6) V(Tap_inQ7) V(Tap_inQ8) V(Tap_inQ9) V(Tap_inQ10) V(Tap_inQ11)
            V(Tap_inQ12) V(Tap_inQ13) V(Tap_inQ14) V(Tap_inQ15)
.print tran "D:\Chris\Thesis\Schematics\testfiles\Tapline\2Tap_Test\16Adder2Inputs2.out"
            V(Tap_inI0) V(Tap_inI1) V(Tap_inI2) V(Tap_inI3) V(Tap_inI4) V(Tap_inI5)
            V(Tap_inI6) V(Tap_inI7) V(Tap_inI8) V(Tap_inI9) V(Tap_inI10) V(Tap_inI11)
            V(Tap_inI12) V(Tap_inI13) V(Tap_inI14) V(Tap_inI15)
.print tran "D:\Chris\Thesis\Schematics\testfiles\Tapline\2Tap_Test\QOUTPUTS2.out"
            V(Tap_outQ0)V(Tap_outQ1) V(Tap_outQ2) V(Tap_outQ3) V(Tap_outQ4)
            V(Tap_outQ5) V(Tap_outQ6) V(Tap_outQ7) V(Tap_outQ8) V(Tap_outQ9)
            V(Tap_outQ10) V(Tap_outQ11) V(Tap_outQ12) V(Tap_outQ13) V(Tap_outQ14)
            V(Tap_outQ15)
.print tran "D:\Chris\Thesis\Schematics\testfiles\Tapline\2Tap_Test\IOUTPUTS2.out"
            V(Tap_outI0)V(Tap_outI1) V(Tap_outI2) V(Tap_outI3) V(Tap_outI4) V(Tap_outI5)
            V(Tap_outI6) V(Tap_outI7) V(Tap_outI8) V(Tap_outI9) V(Tap_outI10)
            V(Tap_outI11) V(Tap_outI12) V(Tap_outI13) V(Tap_outI14) V(Tap_outI15)
.print tran "D:\Chris\Thesis\Schematics\testfiles\Tapline\2Tap_Test\CTRLOutputs2.out"
            V(valid_result_out) V(Overflow_out) V(S_P_Test_Lout) V(S_P_Test_Rout)
.param l=0.05u
.include "D:\Chris\Thesis\ModelParammod.md"
```

```
*
* Waveform probing commands
.probe
.options probefilename="D:\Chris\Thesis\Schematics\Testfiles\DISNov16.dat"
+ probesdbfile="D:\Chris\Thesis\Schematics\DISNov16.sdb"
+ probetopmodule="2-Tapline"

* No Ports in cell: PageID_Tanner
* End of module with no ports: PageID_Tanner

.SUBCKT Inv A Out Gnd Vdd
M2 Out A Gnd Gnd NMOS W='28*l' L='7*l' AS='148*l*l' AD='144*l*l' PS='68*l' PD='68*l'
M=1
* Page Size:  5x7
* S-Edit  Inverter (TIB)
* Designed by: J. Luo  Aug 19, 1999  09:04:52
* Schematic generated by S-Edit
* from file D:\Chris\Thesis\Schematics\DISNov16 / module Inv / page Page0
M1 Out A Vdd Vdd PMOS W='28*l' L='7*l' AS='148*l*l' AD='144*l*l' PS='68*l' PD='68*l'
M=1
.ENDS

.SUBCKT NOR2 A B Out Gnd Vdd
M4 Out B Gnd Gnd NMOS W='14*l' L='7*l' AS='144*l*l' AD='84*l*l' PS='68*l' PD='34*l'
M=1
M3 Out A Gnd Gnd NMOS W='14*l' L='7*l' AS='84*l*l' AD='144*l*l' PS='34*l' PD='68*l'
M=1
* Page Size:  5x7
* S-Edit  2-Input NOR Gate (TIB)
* Designed by: J. Luo  Oct 6, 1999  19:12:33
* Schematic generated by S-Edit
* from file D:\Chris\Thesis\Schematics\DISNov16 / module NOR2 / page Page0
M2 Out B 1 Vdd PMOS W='28*l' L='7*l' AS='148*l*l' AD='84*l*l' PS='68*l' PD='34*l' M=1
M1 1 A Vdd Vdd PMOS W='28*l' L='7*l' AS='84*l*l' AD='144*l*l' PS='34*l' PD='68*l' M=1
.ENDS

.SUBCKT OR2 A B Out Gnd Vdd
XInv_1 N4 Out Gnd Vdd Inv
XNOR2_1 A B N4 Gnd Vdd NOR2
.ENDS

.SUBCKT NOR3 A B C Out Gnd Vdd
M4 Out A Gnd Gnd NMOS W='14*l' L='7*l' AS='84*l*l' AD='144*l*l' PS='34*l' PD='68*l'
M=1
M5 Out B Gnd Gnd NMOS W='14*l' L='7*l' AS='84*l*l' AD='84*l*l' PS='34*l' PD='34*l' M=1
M6 Out C Gnd Gnd NMOS W='14*l' L='7*l' AS='148*l*l' AD='84*l*l' PS='68*l' PD='34*l'
M=1
* Page Size:  5x7
* S-Edit  3-Input NOR Gate (TIB)
```

```
* Designed by: J. Luo  Oct 6, 1999  19:13:24
* Schematic generated by S-Edit
* from file D:\Chris\Thesis\Schematics\DISNov16 / module NOR3 / page Page0
M1 1 A Vdd Vdd PMOS W='28*l' L='7*l' AS='84*l*l' AD='144*l*l' PS='34*l' PD='68*l' M=1
M2 2 B 1 Vdd PMOS W='28*l' L='7*l' AS='84*l*l' AD='84*l*l' PS='34*l' PD='34*l' M=1
M3 Out C 2 Vdd PMOS W='28*l' L='7*l' AS='148*l*l' AD='84*l*l' PS='68*l' PD='34*l' M=1
.ENDS

* No Ports in cell: Legend
* End of module with no ports: Legend

.SUBCKT 5-to-32_Decoder_part1 A0 A1 A01 A2 A02 A3 A4 A11 A12 A21 A22 A31 A32
+ A41 A42 inv_A01 inv_A02 inv_A11 inv_A12 inv_A21 inv_A22 inv_A31 inv_A32 inv_A41
+ inv_A42 Gnd Vdd
XInv_1 N2 A42 Gnd Vdd Inv
XInv_2 N2 A41 Gnd Vdd Inv
XInv_3 A4 inv_A42 Gnd Vdd Inv
XInv_4 A4 inv_A41 Gnd Vdd Inv
XInv_5 N3 A32 Gnd Vdd Inv
XInv_6 N3 A31 Gnd Vdd Inv
XInv_7 A3 inv_A32 Gnd Vdd Inv
XInv_8 A3 inv_A31 Gnd Vdd Inv
XInv_9 N9 A22 Gnd Vdd Inv
XInv_10 N9 A21 Gnd Vdd Inv
XInv_11 A2 inv_A22 Gnd Vdd Inv
XInv_12 A2 inv_A21 Gnd Vdd Inv
XInv_13 N15 A12 Gnd Vdd Inv
XInv_14 N15 A11 Gnd Vdd Inv
XInv_15 A1 inv_A12 Gnd Vdd Inv
XInv_16 A1 inv_A11 Gnd Vdd Inv
XInv_17 N19 A01 Gnd Vdd Inv
XInv_18 N19 A02 Gnd Vdd Inv
XInv_19 A0 inv_A02 Gnd Vdd Inv
XInv_20 A0 inv_A01 Gnd Vdd Inv
XInv_21 A4 N2 Gnd Vdd Inv
XInv_22 A3 N3 Gnd Vdd Inv
XInv_23 A1 N15 Gnd Vdd Inv
XInv_24 A2 N9 Gnd Vdd Inv
XInv_25 A0 N19 Gnd Vdd Inv
.ENDS

.SUBCKT NAND5 A B C D E Out Gnd Vdd
M5 Out E N2 Gnd NMOS W='14*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
M6 N2 D N4 Gnd NMOS W='14*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
M7 N4 C N1 Gnd NMOS W='14*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
M8 N1 B N5 Gnd NMOS W='14*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
M9 N5 A Gnd Gnd NMOS W='14*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
M0 Out E Vdd Vdd PMOS W='28*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
M1 Out D Vdd Vdd PMOS W='28*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
```

```
M2 Out C Vdd Vdd PMOS W='28*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
M3 Out B Vdd Vdd PMOS W='28*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
M4 Out A Vdd Vdd PMOS W='28*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
.ENDS

.SUBCKT 5-to-32_Decoder_part2 A01 A02 A11 A12 A21 A22 A31 A32 A41 A42 inv_A01
+ inv_A02 inv_A11 inv_A12 inv_A21 inv_A22 inv_A31 inv_A32 inv_A41 inv_A42 inv_W0
+ inv_W1 inv_W2 inv_W3 inv_W4 inv_W5 inv_W6 inv_W7 inv_W8 inv_W9 inv_W10
inv_W11
+ inv_W12 inv_W13 inv_W14 inv_W15 inv_W16 inv_W17 inv_W18 inv_W19 inv_W20
inv_W21
+ inv_W22 inv_W23 inv_W24 inv_W25 inv_W26 inv_W27 inv_W28 inv_W29 inv_W30
inv_W31
+ W0 W1 W2 W3 W4 W5 W6 W7 W8 W9 W10 W11 W12 W13 W14 W15 W16 W17 W18
W19 W20 W21
+ W22 W23 W24 W25 W26 W27 W28 W29 W30 W31 Gnd Vdd
XInv_1 inv_W1 W1 Gnd Vdd Inv
XInv_2 inv_W0 W0 Gnd Vdd Inv
XInv_3 inv_W2 W2 Gnd Vdd Inv
XInv_4 inv_W3 W3 Gnd Vdd Inv
XInv_5 inv_W4 W4 Gnd Vdd Inv
XInv_6 inv_W5 W5 Gnd Vdd Inv
XInv_7 inv_W6 W6 Gnd Vdd Inv
XInv_8 inv_W7 W7 Gnd Vdd Inv
XInv_10 inv_W8 W8 Gnd Vdd Inv
XInv_11 inv_W9 W9 Gnd Vdd Inv
XInv_12 inv_W10 W10 Gnd Vdd Inv
XInv_13 inv_W11 W11 Gnd Vdd Inv
XInv_14 inv_W15 W15 Gnd Vdd Inv
XInv_15 inv_W14 W14 Gnd Vdd Inv
XInv_16 inv_W13 W13 Gnd Vdd Inv
XInv_17 inv_W12 W12 Gnd Vdd Inv
XInv_18 inv_W19 W19 Gnd Vdd Inv
XInv_19 inv_W18 W18 Gnd Vdd Inv
XInv_20 inv_W17 W17 Gnd Vdd Inv
XInv_21 inv_W16 W16 Gnd Vdd Inv
XInv_22 inv_W23 W23 Gnd Vdd Inv
XInv_23 inv_W22 W22 Gnd Vdd Inv
XInv_24 inv_W21 W21 Gnd Vdd Inv
XInv_25 inv_W20 W20 Gnd Vdd Inv
XInv_26 inv_W27 W27 Gnd Vdd Inv
XInv_27 inv_W26 W26 Gnd Vdd Inv
XInv_28 inv_W25 W25 Gnd Vdd Inv
XInv_29 inv_W24 W24 Gnd Vdd Inv
XInv_30 inv_W31 W31 Gnd Vdd Inv
XInv_31 inv_W30 W30 Gnd Vdd Inv
XInv_32 inv_W29 W29 Gnd Vdd Inv
XInv_33 inv_W28 W28 Gnd Vdd Inv
XNAND5_1 inv_A41 inv_A31 inv_A21 inv_A11 inv_A01 inv_W0 Gnd Vdd NAND5
```

```
XNAND5_2 inv_A41 inv_A31 inv_A21 inv_A11 A01 inv_W1 Gnd Vdd NAND5
XNAND5_3 inv_A41 inv_A31 inv_A21 A11 inv_A01 inv_W2 Gnd Vdd NAND5
XNAND5_4 inv_A41 inv_A31 inv_A21 A11 A01 inv_W3 Gnd Vdd NAND5
XNAND5_5 inv_A41 inv_A31 A21 inv_A11 inv_A01 inv_W4 Gnd Vdd NAND5
XNAND5_6 inv_A41 inv_A31 A21 inv_A11 A01 inv_W5 Gnd Vdd NAND5
XNAND5_7 inv_A41 inv_A31 A21 A11 inv_A01 inv_W6 Gnd Vdd NAND5
XNAND5_8 inv_A41 inv_A31 A21 A11 A01 inv_W7 Gnd Vdd NAND5
XNAND5_9 inv_A42 A31 inv_A21 inv_A11 inv_A01 inv_W8 Gnd Vdd NAND5
XNAND5_10 inv_A42 A31 inv_A21 inv_A11 A01 inv_W9 Gnd Vdd NAND5
XNAND5_11 inv_A42 A31 inv_A21 A11 inv_A01 inv_W10 Gnd Vdd NAND5
XNAND5_12 inv_A42 A31 inv_A21 A11 A01 inv_W11 Gnd Vdd NAND5
XNAND5_13 inv_A42 A31 A21 inv_A11 inv_A01 inv_W12 Gnd Vdd NAND5
XNAND5_14 inv_A42 A31 A21 inv_A11 A01 inv_W13 Gnd Vdd NAND5
XNAND5_15 inv_A42 A31 A21 A11 inv_A01 inv_W14 Gnd Vdd NAND5
XNAND5_16 inv_A42 A31 A21 A11 A01 inv_W15 Gnd Vdd NAND5
XNAND5_17 A41 inv_A32 inv_A22 inv_A12 inv_A02 inv_W16 Gnd Vdd NAND5
XNAND5_18 A41 inv_A32 inv_A22 inv_A12 A02 inv_W17 Gnd Vdd NAND5
XNAND5_19 A41 inv_A32 inv_A22 A12 inv_A02 inv_W18 Gnd Vdd NAND5
XNAND5_20 A41 inv_A32 inv_A22 A12 A02 inv_W19 Gnd Vdd NAND5
XNAND5_21 A41 inv_A32 A22 inv_A12 inv_A02 inv_W20 Gnd Vdd NAND5
XNAND5_22 A41 inv_A32 A22 inv_A12 A02 inv_W21 Gnd Vdd NAND5
XNAND5_23 A41 inv_A32 A22 A12 inv_A02 inv_W22 Gnd Vdd NAND5
XNAND5_24 A41 inv_A32 A22 A12 A02 inv_W23 Gnd Vdd NAND5
XNAND5_25 A42 A32 inv_A22 inv_A12 inv_A02 inv_W24 Gnd Vdd NAND5
XNAND5_26 A42 A32 inv_A22 inv_A12 A02 inv_W25 Gnd Vdd NAND5
XNAND5_27 A42 A32 inv_A22 A12 inv_A02 inv_W26 Gnd Vdd NAND5
XNAND5_28 A42 A32 inv_A22 A12 A02 inv_W27 Gnd Vdd NAND5
XNAND5_29 A42 A32 A22 inv_A12 inv_A02 inv_W28 Gnd Vdd NAND5
XNAND5_30 A42 A32 A22 inv_A12 A02 inv_W29 Gnd Vdd NAND5
XNAND5_31 A42 A32 A22 A12 inv_A02 inv_W30 Gnd Vdd NAND5
XNAND5_32 A42 A32 A22 A12 A02 inv_W31 Gnd Vdd NAND5
.ENDS

.SUBCKT N_3 D G S Gnd
M0 D G S Gnd NMOS W='14*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
.ENDS

.SUBCKT P_3 D G S Vdd
M0 D G S Vdd PMOS W='28*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
.ENDS

.SUBCKT LUT_ROM_* SPICE netlist written by S-Edit Win32 6.00
* Written on Nov 19, 1999 at 13:57:23
Vdd Vdd Gnd DC 5
.include "D:\Chris\Thesis\schematics\testfiles\Tapline\2Tap_Test\input_tableTapTest.md"
*
.options  abstol=50n reltol=0.001 numnt=6 prtdel=400n
.tran 300n 68000n start=390n
```

```
.print tran "D:\Chris\Thesis\Schematics\testfiles\Tapline\2Tap_Test\CTRLInputs2.out" V(CLK)
V(SL) V(SR) V(LD) V(HLD) V(delta_Phase_inc) V(Phase0) V(Phase1) V(Phase2) V(Phase3)
V(Phase4) V(LD_Phase_inc) V(Tgt_Extent0) V(Tgt_Extent1) V(Range_bin_valid)
V(S_P_Test_Rin) V(S_P_Test_Lin)
.print tran "D:\Chris\Thesis\Schematics\testfiles\Tapline\2Tap_Test\Inputs2.out"
V(Phase_inc0_Tap0) V(Phase_inc1_Tap0) V(Phase_inc2_Tap0) V(Phase_inc3_Tap0)
V(Phase_inc0_Tap1) V(Phase_inc1_Tap1) V(Phase_inc2_Tap1) V(Phase_inc3_Tap1)
V(LD_Gain_Reg) V(Gain_Dat0_Tap0) V(Gain_Dat1_Tap0) V(Gain_Dat0_Tap1)
V(Gain_Dat1_Tap1)
.print tran "D:\Chris\Thesis\Schematics\testfiles\Tapline\2Tap_Test\16Adder1Inputs2.out"
V(Tap_inQ0) V(Tap_inQ1) V(Tap_inQ2) V(Tap_inQ3) V(Tap_inQ4) V(Tap_inQ5)
V(Tap_inQ6) V(Tap_inQ7) V(Tap_inQ8) V(Tap_inQ9) V(Tap_inQ10) V(Tap_inQ11)
V(Tap_inQ12) V(Tap_inQ13) V(Tap_inQ14) V(Tap_inQ15)
.print tran "D:\Chris\Thesis\Schematics\testfiles\Tapline\2Tap_Test\16Adder2Inputs2.out"
V(Tap_inI0) V(Tap_inI1) V(Tap_inI2) V(Tap_inI3) V(Tap_inI4) V(Tap_inI5) V(Tap_inI6)
V(Tap_inI7) V(Tap_inI8) V(Tap_inI9) V(Tap_inI10) V(Tap_inI11) V(Tap_inI12) V(Tap_inI13)
V(Tap_inI14) V(Tap_inI15)
*
.print tran "D:\Chris\Thesis\Schematics\testfiles\Tapline\2Tap_Test\QOUTPUTS2.out"
V(Tap_outQ0)V(Tap_outQ1) V(Tap_outQ2) V(Tap_outQ3) V(Tap_outQ4) V(Tap_outQ5)
V(Tap_outQ6) V(Tap_outQ7) V(Tap_outQ8) V(Tap_outQ9) V(Tap_outQ10) V(Tap_outQ11)
V(Tap_outQ12) V(Tap_outQ13) V(Tap_outQ14) V(Tap_outQ15)
.print tran "D:\Chris\Thesis\Schematics\testfiles\Tapline\2Tap_Test\IOUTPUTS2.out"
V(Tap_outI0)V(Tap_outI1) V(Tap_outI2) V(Tap_outI3) V(Tap_outI4) V(Tap_outI5)
V(Tap_outI6) V(Tap_outI7) V(Tap_outI8) V(Tap_outI9) V(Tap_outI10) V(Tap_outI11)
V(Tap_outI12) V(Tap_outI13) V(Tap_outI14) V(Tap_outI15)
*
.print tran "D:\Chris\Thesis\Schematics\testfiles\Tapline\2Tap_Test\CTRLOutputs2.out"
V(valid_result_out) V(Overflow_out) V(S_P_Test_Lout) V(S_P_Test_Rout)
*
.param l=0.05u
*
.include "D:\Chris\Thesis\ModelParammod.md"
*
*
* Waveform probing commands
.probe
.options probefilename="D:\Chris\Thesis\Schematics\Testfiles\DISNov16.dat"
+ probesdbfile="D:\Chris\Thesis\Schematics\DISNov16.sdb"
+ probetopmodule="2-Tapline"

* No Ports in cell: PageID_Tanner
* End of module with no ports: PageID_Tanner

.SUBCKT Inv A Out Gnd Vdd
M2 Out A Gnd Gnd NMOS W='28*l' L='7*l' AS='148*l*l' AD='144*l*l' PS='68*l' PD='68*l'
M=1
* Page Size:  5x7
* S-Edit  Inverter (TIB)
```

```
* Designed by: J. Luo  Aug 19, 1999  09:04:52
* Schematic generated by S-Edit
* from file D:\Chris\Thesis\Schematics\DISNov16 / module Inv / page Page0
M1 Out A Vdd Vdd PMOS W='28*l' L='7*l' AS='148*l*l' AD='144*l*l' PS='68*l' PD='68*l'
M=1
.ENDS

.SUBCKT NOR2 A B Out Gnd Vdd
M4 Out B Gnd Gnd NMOS W='14*l' L='7*l' AS='144*l*l' AD='84*l*l' PS='68*l' PD='34*l'
M=1
M3 Out A Gnd Gnd NMOS W='14*l' L='7*l' AS='84*l*l' AD='144*l*l' PS='34*l' PD='68*l'
M=1
* Page Size:  5x7
* S-Edit  2-Input NOR Gate (TIB)
* Designed by: J. Luo  Oct 6, 1999  19:12:33
* Schematic generated by S-Edit
* from file D:\Chris\Thesis\Schematics\DISNov16 / module NOR2 / page Page0
M2 Out B 1 Vdd PMOS W='28*l' L='7*l' AS='148*l*l' AD='84*l*l' PS='68*l' PD='34*l' M=1
M1 1 A Vdd Vdd PMOS W='28*l' L='7*l' AS='84*l*l' AD='144*l*l' PS='34*l' PD='68*l' M=1
.ENDS

.SUBCKT OR2 A B Out Gnd Vdd
XInv_1 N4 Out Gnd Vdd Inv
XNOR2_1 A B N4 Gnd Vdd NOR2
.ENDS

.SUBCKT NOR3 A B C Out Gnd Vdd
M4 Out A Gnd Gnd NMOS W='14*l' L='7*l' AS='84*l*l' AD='144*l*l' PS='34*l' PD='68*l'
M=1
M5 Out B Gnd Gnd NMOS W='14*l' L='7*l' AS='84*l*l' AD='84*l*l' PS='34*l' PD='34*l' M=1
M6 Out C Gnd Gnd NMOS W='14*l' L='7*l' AS='148*l*l' AD='84*l*l' PS='68*l' PD='34*l'
M=1
* Page Size:  5x7
* S-Edit  3-Input NOR Gate (TIB)
* Designed by: J. Luo  Oct 6, 1999  19:13:24
* Schematic generated by S-Edit
* from file D:\Chris\Thesis\Schematics\DISNov16 / module NOR3 / page Page0
M1 1 A Vdd Vdd PMOS W='28*l' L='7*l' AS='84*l*l' AD='144*l*l' PS='34*l' PD='68*l' M=1
M2 2 B 1 Vdd PMOS W='28*l' L='7*l' AS='84*l*l' AD='84*l*l' PS='34*l' PD='34*l' M=1
M3 Out C 2 Vdd PMOS W='28*l' L='7*l' AS='148*l*l' AD='84*l*l' PS='68*l' PD='34*l' M=1
.ENDS

* No Ports in cell: Legend
* End of module with no ports: Legend

.SUBCKT 5-to-32_Decoder_part1 A0 A1 A01 A2 A02 A3 A4 A11 A12 A21 A22 A31 A32
+ A41 A42 inv_A01 inv_A02 inv_A11 inv_A12 inv_A21 inv_A22 inv_A31 inv_A32 inv_A41
+ inv_A42 Gnd Vdd
XInv_1 N2 A42 Gnd Vdd Inv
```

```
XInv_2 N2 A41 Gnd Vdd Inv
XInv_3 A4 inv_A42 Gnd Vdd Inv
XInv_4 A4 inv_A41 Gnd Vdd Inv
XInv_5 N3 A32 Gnd Vdd Inv
XInv_6 N3 A31 Gnd Vdd Inv
XInv_7 A3 inv_A32 Gnd Vdd Inv
XInv_8 A3 inv_A31 Gnd Vdd Inv
XInv_9 N9 A22 Gnd Vdd Inv
XInv_10 N9 A21 Gnd Vdd Inv
XInv_11 A2 inv_A22 Gnd Vdd Inv
XInv_12 A2 inv_A21 Gnd Vdd Inv
XInv_13 N15 A12 Gnd Vdd Inv
XInv_14 N15 A11 Gnd Vdd Inv
XInv_15 A1 inv_A12 Gnd Vdd Inv
XInv_16 A1 inv_A11 Gnd Vdd Inv
XInv_17 N19 A01 Gnd Vdd Inv
XInv_18 N19 A02 Gnd Vdd Inv
XInv_19 A0 inv_A02 Gnd Vdd Inv
XInv_20 A0 inv_A01 Gnd Vdd Inv
XInv_21 A4 N2 Gnd Vdd Inv
XInv_22 A3 N3 Gnd Vdd Inv
XInv_23 A1 N15 Gnd Vdd Inv
XInv_24 A2 N9 Gnd Vdd Inv
XInv_25 A0 N19 Gnd Vdd Inv
.ENDS

.SUBCKT NAND5 A B C D E Out Gnd Vdd
M5 Out E N2 Gnd NMOS W='14*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
M6 N2 D N4 Gnd NMOS W='14*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
M7 N4 C N1 Gnd NMOS W='14*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
M8 N1 B N5 Gnd NMOS W='14*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
M9 N5 A Gnd Gnd NMOS W='14*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
M0 Out E Vdd Vdd PMOS W='28*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
M1 Out D Vdd Vdd PMOS W='28*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
M2 Out C Vdd Vdd PMOS W='28*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
M3 Out B Vdd Vdd PMOS W='28*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
M4 Out A Vdd Vdd PMOS W='28*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
.ENDS

.SUBCKT 5-to-32_Decoder_part2 A01 A02 A11 A12 A21 A22 A31 A32 A41 A42 inv_A01
+ inv_A02 inv_A11 inv_A12 inv_A21 inv_A22 inv_A31 inv_A32 inv_A41 inv_A42 inv_W0
+ inv_W1 inv_W2 inv_W3 inv_W4 inv_W5 inv_W6 inv_W7 inv_W8 inv_W9 inv_W10
inv_W11
+ inv_W12 inv_W13 inv_W14 inv_W15 inv_W16 inv_W17 inv_W18 inv_W19 inv_W20
inv_W21
+ inv_W22 inv_W23 inv_W24 inv_W25 inv_W26 inv_W27 inv_W28 inv_W29 inv_W30
inv_W31
+ W0 W1 W2 W3 W4 W5 W6 W7 W8 W9 W10 W11 W12 W13 W14 W15 W16 W17 W18
W19 W20 W21
```

```
+ W22 W23 W24 W25 W26 W27 W28 W29 W30 W31 Gnd Vdd
XInv_1 inv_W1 W1 Gnd Vdd Inv
XInv_2 inv_W0 W0 Gnd Vdd Inv
XInv_3 inv_W2 W2 Gnd Vdd Inv
XInv_4 inv_W3 W3 Gnd Vdd Inv
XInv_5 inv_W4 W4 Gnd Vdd Inv
XInv_6 inv_W5 W5 Gnd Vdd Inv
XInv_7 inv_W6 W6 Gnd Vdd Inv
XInv_8 inv_W7 W7 Gnd Vdd Inv
XInv_10 inv_W8 W8 Gnd Vdd Inv
XInv_11 inv_W9 W9 Gnd Vdd Inv
XInv_12 inv_W10 W10 Gnd Vdd Inv
XInv_13 inv_W11 W11 Gnd Vdd Inv
XInv_14 inv_W15 W15 Gnd Vdd Inv
XInv_15 inv_W14 W14 Gnd Vdd Inv
XInv_16 inv_W13 W13 Gnd Vdd Inv
XInv_17 inv_W12 W12 Gnd Vdd Inv
XInv_18 inv_W19 W19 Gnd Vdd Inv
XInv_19 inv_W18 W18 Gnd Vdd Inv
XInv_20 inv_W17 W17 Gnd Vdd Inv
XInv_21 inv_W16 W16 Gnd Vdd Inv
XInv_22 inv_W23 W23 Gnd Vdd Inv
XInv_23 inv_W22 W22 Gnd Vdd Inv
XInv_24 inv_W21 W21 Gnd Vdd Inv
XInv_25 inv_W20 W20 Gnd Vdd Inv
XInv_26 inv_W27 W27 Gnd Vdd Inv
XInv_27 inv_W26 W26 Gnd Vdd Inv
XInv_28 inv_W25 W25 Gnd Vdd Inv
XInv_29 inv_W24 W24 Gnd Vdd Inv
XInv_30 inv_W31 W31 Gnd Vdd Inv
XInv_31 inv_W30 W30 Gnd Vdd Inv
XInv_32 inv_W29 W29 Gnd Vdd Inv
XInv_33 inv_W28 W28 Gnd Vdd Inv
XNAND5_1 inv_A41 inv_A31 inv_A21 inv_A11 inv_A01 inv_W0 Gnd Vdd NAND5
XNAND5_2 inv_A41 inv_A31 inv_A21 inv_A11 A01 inv_W1 Gnd Vdd NAND5
XNAND5_3 inv_A41 inv_A31 inv_A21 A11 inv_A01 inv_W2 Gnd Vdd NAND5
XNAND5_4 inv_A41 inv_A31 inv_A21 A11 A01 inv_W3 Gnd Vdd NAND5
XNAND5_5 inv_A41 inv_A31 A21 inv_A11 inv_A01 inv_W4 Gnd Vdd NAND5
XNAND5_6 inv_A41 inv_A31 A21 inv_A11 A01 inv_W5 Gnd Vdd NAND5
XNAND5_7 inv_A41 inv_A31 A21 A11 inv_A01 inv_W6 Gnd Vdd NAND5
XNAND5_8 inv_A41 inv_A31 A21 A11 A01 inv_W7 Gnd Vdd NAND5
XNAND5_9 inv_A42 A31 inv_A21 inv_A11 inv_A01 inv_W8 Gnd Vdd NAND5
XNAND5_10 inv_A42 A31 inv_A21 inv_A11 A01 inv_W9 Gnd Vdd NAND5
XNAND5_11 inv_A42 A31 inv_A21 A11 inv_A01 inv_W10 Gnd Vdd NAND5
XNAND5_12 inv_A42 A31 inv_A21 A11 A01 inv_W11 Gnd Vdd NAND5
XNAND5_13 inv_A42 A31 A21 inv_A11 inv_A01 inv_W12 Gnd Vdd NAND5
XNAND5_14 inv_A42 A31 A21 inv_A11 A01 inv_W13 Gnd Vdd NAND5
XNAND5_15 inv_A42 A31 A21 A11 inv_A01 inv_W14 Gnd Vdd NAND5
XNAND5_16 inv_A42 A31 A21 A11 A01 inv_W15 Gnd Vdd NAND5
```

```
XNAND5_17 A41 inv_A32 inv_A22 inv_A12 inv_A02 inv_W16 Gnd Vdd NAND5
XNAND5_18 A41 inv_A32 inv_A22 inv_A12 A02 inv_W17 Gnd Vdd NAND5
XNAND5_19 A41 inv_A32 inv_A22 A12 inv_A02 inv_W18 Gnd Vdd NAND5
XNAND5_20 A41 inv_A32 inv_A22 A12 A02 inv_W19 Gnd Vdd NAND5
XNAND5_21 A41 inv_A32 A22 inv_A12 inv_A02 inv_W20 Gnd Vdd NAND5
XNAND5_22 A41 inv_A32 A22 inv_A12 A02 inv_W21 Gnd Vdd NAND5
XNAND5_23 A41 inv_A32 A22 A12 inv_A02 inv_W22 Gnd Vdd NAND5
XNAND5_24 A41 inv_A32 A22 A12 A02 inv_W23 Gnd Vdd NAND5
XNAND5_25 A42 A32 inv_A22 inv_A12 inv_A02 inv_W24 Gnd Vdd NAND5
XNAND5_26 A42 A32 inv_A22 inv_A12 A02 inv_W25 Gnd Vdd NAND5
XNAND5_27 A42 A32 inv_A22 A12 inv_A02 inv_W26 Gnd Vdd NAND5
XNAND5_28 A42 A32 inv_A22 A12 A02 inv_W27 Gnd Vdd NAND5
XNAND5_29 A42 A32 A22 inv_A12 inv_A02 inv_W28 Gnd Vdd NAND5
XNAND5_30 A42 A32 A22 inv_A12 A02 inv_W29 Gnd Vdd NAND5
XNAND5_31 A42 A32 A22 A12 inv_A02 inv_W30 Gnd Vdd NAND5
XNAND5_32 A42 A32 A22 A12 A02 inv_W31 Gnd Vdd NAND5
.ENDS

.SUBCKT N_3 D G S Gnd
M0 D G S Gnd NMOS W='14*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
.ENDS

.SUBCKT P_3 D G S Vdd
M0 D G S Vdd PMOS W='28*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
.ENDS

.SUBCKT LUT_ROM_* SPICE netlist written by S-Edit Win32 6.00
* Written on Nov 19, 1999 at 13:57:23
Vdd Vdd Gnd DC 5
.include "D:\Chris\Thesis\schematics\testfiles\Tapline\2Tap_Test\input_tableTapTest.md"
*
.options  abstol=50n reltol=0.001 numnt=6 prtdel=400n
.tran 300n 68000n start=390n
.print tran "D:\Chris\Thesis\Schematics\testfiles\Tapline\2Tap_Test\CTRLInputs2.out" V(CLK)
V(SL) V(SR) V(LD) V(HLD) V(delta_Phase_inc) V(Phase0) V(Phase1) V(Phase2) V(Phase3)
V(Phase4) V(LD_Phase_inc) V(Tgt_Extent0) V(Tgt_Extent1) V(Range_bin_valid)
V(S_P_Test_Rin) V(S_P_Test_Lin)
.print tran "D:\Chris\Thesis\Schematics\testfiles\Tapline\2Tap_Test\Inputs2.out"
V(Phase_inc0_Tap0) V(Phase_inc1_Tap0) V(Phase_inc2_Tap0) V(Phase_inc3_Tap0)
V(Phase_inc0_Tap1) V(Phase_inc1_Tap1) V(Phase_inc2_Tap1) V(Phase_inc3_Tap1)
V(LD_Gain_Reg) V(Gain_Dat0_Tap0) V(Gain_Dat1_Tap0) V(Gain_Dat0_Tap1)
V(Gain_Dat1_Tap1)
.print tran "D:\Chris\Thesis\Schematics\testfiles\Tapline\2Tap_Test\16Adder1Inputs2.out"
V(Tap_inQ0) V(Tap_inQ1) V(Tap_inQ2) V(Tap_inQ3) V(Tap_inQ4) V(Tap_inQ5)
V(Tap_inQ6) V(Tap_inQ7) V(Tap_inQ8) V(Tap_inQ9) V(Tap_inQ10) V(Tap_inQ11)
V(Tap_inQ12) V(Tap_inQ13) V(Tap_inQ14) V(Tap_inQ15)
.print tran "D:\Chris\Thesis\Schematics\testfiles\Tapline\2Tap_Test\16Adder2Inputs2.out"
V(Tap_inI0) V(Tap_inI1) V(Tap_inI2) V(Tap_inI3) V(Tap_inI4) V(Tap_inI5) V(Tap_inI6)
```

```
V(Tap_inI7) V(Tap_inI8) V(Tap_inI9) V(Tap_inI10) V(Tap_inI11) V(Tap_inI12) V(Tap_inI13)
V(Tap_inI14) V(Tap_inI15)
*
.print tran "D:\Chris\Thesis\Schematics\testfiles\Tapline\2Tap_Test\QOUTPUTS2.out"
V(Tap_outQ0)V(Tap_outQ1) V(Tap_outQ2) V(Tap_outQ3) V(Tap_outQ4) V(Tap_outQ5)
V(Tap_outQ6) V(Tap_outQ7) V(Tap_outQ8) V(Tap_outQ9) V(Tap_outQ10) V(Tap_outQ11)
V(Tap_outQ12) V(Tap_outQ13) V(Tap_outQ14) V(Tap_outQ15)
.print tran "D:\Chris\Thesis\Schematics\testfiles\Tapline\2Tap_Test\IOUTPUTS2.out"
V(Tap_outI0)V(Tap_outI1) V(Tap_outI2) V(Tap_outI3) V(Tap_outI4) V(Tap_outI5)
V(Tap_outI6) V(Tap_outI7) V(Tap_outI8) V(Tap_outI9) V(Tap_outI10) V(Tap_outI11)
V(Tap_outI12) V(Tap_outI13) V(Tap_outI14) V(Tap_outI15)
*
.print tran "D:\Chris\Thesis\Schematics\testfiles\Tapline\2Tap_Test\CTRLOutputs2.out"
V(valid_result_out) V(Overflow_out) V(S_P_Test_Lout) V(S_P_Test_Rout)
*
.param l=0.05u
*
.include "D:\Chris\Thesis\ModelParammod.md"
*
*
* Waveform probing commands
.probe
.options probefilename="D:\Chris\Thesis\Schematics\Testfiles\DISNov16.dat"
+ probesdbfile="D:\Chris\Thesis\Schematics\DISNov16.sdb"
+ probetopmodule="2-Tapline"

* No Ports in cell: PageID_Tanner
* End of module with no ports: PageID_Tanner

.SUBCKT Inv A Out Gnd Vdd
M2 Out A Gnd Gnd NMOS W='28*l' L='7*l' AS='148*l*l' AD='144*l*l' PS='68*l' PD='68*l'
M=1
* Page Size:  5x7
* S-Edit  Inverter (TIB)
* Designed by: J. Luo  Aug 19, 1999  09:04:52
* Schematic generated by S-Edit
* from file D:\Chris\Thesis\Schematics\DISNov16 / module Inv / page Page0
M1 Out A Vdd Vdd PMOS W='28*l' L='7*l' AS='148*l*l' AD='144*l*l' PS='68*l' PD='68*l'
M=1
.ENDS

.SUBCKT NOR2 A B Out Gnd Vdd
M4 Out B Gnd Gnd NMOS W='14*l' L='7*l' AS='144*l*l' AD='84*l*l' PS='68*l' PD='34*l'
M=1
M3 Out A Gnd Gnd NMOS W='14*l' L='7*l' AS='84*l*l' AD='144*l*l' PS='34*l' PD='68*l'
M=1
* Page Size:  5x7
* S-Edit  2-Input NOR Gate (TIB)
* Designed by: J. Luo  Oct 6, 1999  19:12:33
```

```
* Schematic generated by S-Edit
* from file D:\Chris\Thesis\Schematics\DISNov16 / module NOR2 / page Page0
M2 Out B 1 Vdd PMOS W='28*l' L='7*l' AS='148*l*l' AD='84*l*l' PS='68*l' PD='34*l' M=1
M1 1 A Vdd Vdd PMOS W='28*l' L='7*l' AS='84*l*l' AD='144*l*l' PS='34*l' PD='68*l' M=1
.ENDS

.SUBCKT OR2 A B Out Gnd Vdd
XInv_1 N4 Out Gnd Vdd Inv
XNOR2_1 A B N4 Gnd Vdd NOR2
.ENDS

.SUBCKT NOR3 A B C Out Gnd Vdd
M4 Out A Gnd Gnd NMOS W='14*l' L='7*l' AS='84*l*l' AD='144*l*l' PS='34*l' PD='68*l'
M=1
M5 Out B Gnd Gnd NMOS W='14*l' L='7*l' AS='84*l*l' AD='84*l*l' PS='34*l' PD='34*l' M=1
M6 Out C Gnd Gnd NMOS W='14*l' L='7*l' AS='148*l*l' AD='84*l*l' PS='68*l' PD='34*l'
M=1
* Page Size:  5x7
* S-Edit  3-Input NOR Gate (TIB)
* Designed by: J. Luo  Oct 6, 1999  19:13:24
* Schematic generated by S-Edit
* from file D:\Chris\Thesis\Schematics\DISNov16 / module NOR3 / page Page0
M1 1 A Vdd Vdd PMOS W='28*l' L='7*l' AS='84*l*l' AD='144*l*l' PS='34*l' PD='68*l' M=1
M2 2 B 1 Vdd PMOS W='28*l' L='7*l' AS='84*l*l' AD='84*l*l' PS='34*l' PD='34*l' M=1
M3 Out C 2 Vdd PMOS W='28*l' L='7*l' AS='148*l*l' AD='84*l*l' PS='68*l' PD='34*l' M=1
.ENDS

* No Ports in cell: Legend
* End of module with no ports: Legend

.SUBCKT 5-to-32_Decoder_part1 A0 A1 A01 A2 A02 A3 A4 A11 A12 A21 A22 A31 A32
+ A41 A42 inv_A01 inv_A02 inv_A11 inv_A12 inv_A21 inv_A22 inv_A31 inv_A32 inv_A41
+ inv_A42 Gnd Vdd
XInv_1 N2 A42 Gnd Vdd Inv
XInv_2 N2 A41 Gnd Vdd Inv
XInv_3 A4 inv_A42 Gnd Vdd Inv
XInv_4 A4 inv_A41 Gnd Vdd Inv
XInv_5 N3 A32 Gnd Vdd Inv
XInv_6 N3 A31 Gnd Vdd Inv
XInv_7 A3 inv_A32 Gnd Vdd Inv
XInv_8 A3 inv_A31 Gnd Vdd Inv
XInv_9 N9 A22 Gnd Vdd Inv
XInv_10 N9 A21 Gnd Vdd Inv
XInv_11 A2 inv_A22 Gnd Vdd Inv
XInv_12 A2 inv_A21 Gnd Vdd Inv
XInv_13 N15 A12 Gnd Vdd Inv
XInv_14 N15 A11 Gnd Vdd Inv
XInv_15 A1 inv_A12 Gnd Vdd Inv
XInv_16 A1 inv_A11 Gnd Vdd Inv
```

```
XInv_17 N19 A01 Gnd Vdd Inv
XInv_18 N19 A02 Gnd Vdd Inv
XInv_19 A0 inv_A02 Gnd Vdd Inv
XInv_20 A0 inv_A01 Gnd Vdd Inv
XInv_21 A4 N2 Gnd Vdd Inv
XInv_22 A3 N3 Gnd Vdd Inv
XInv_23 A1 N15 Gnd Vdd Inv
XInv_24 A2 N9 Gnd Vdd Inv
XInv_25 A0 N19 Gnd Vdd Inv
.ENDS

.SUBCKT NAND5 A B C D E Out Gnd Vdd
M5 Out E N2 Gnd NMOS W='14*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
M6 N2 D N4 Gnd NMOS W='14*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
M7 N4 C N1 Gnd NMOS W='14*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
M8 N1 B N5 Gnd NMOS W='14*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
M9 N5 A Gnd Gnd NMOS W='14*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
M0 Out E Vdd Vdd PMOS W='28*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
M1 Out D Vdd Vdd PMOS W='28*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
M2 Out C Vdd Vdd PMOS W='28*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
M3 Out B Vdd Vdd PMOS W='28*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
M4 Out A Vdd Vdd PMOS W='28*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
.ENDS

.SUBCKT 5-to-32_Decoder_part2 A01 A02 A11 A12 A21 A22 A31 A32 A41 A42 inv_A01
+ inv_A02 inv_A11 inv_A12 inv_A21 inv_A22 inv_A31 inv_A32 inv_A41 inv_A42 inv_W0
+ inv_W1 inv_W2 inv_W3 inv_W4 inv_W5 inv_W6 inv_W7 inv_W8 inv_W9 inv_W10
inv_W11
+ inv_W12 inv_W13 inv_W14 inv_W15 inv_W16 inv_W17 inv_W18 inv_W19 inv_W20
inv_W21
+ inv_W22 inv_W23 inv_W24 inv_W25 inv_W26 inv_W27 inv_W28 inv_W29 inv_W30
inv_W31
+ W0 W1 W2 W3 W4 W5 W6 W7 W8 W9 W10 W11 W12 W13 W14 W15 W16 W17 W18
W19 W20 W21
+ W22 W23 W24 W25 W26 W27 W28 W29 W30 W31 Gnd Vdd
XInv_1 inv_W1 W1 Gnd Vdd Inv
XInv_2 inv_W0 W0 Gnd Vdd Inv
XInv_3 inv_W2 W2 Gnd Vdd Inv
XInv_4 inv_W3 W3 Gnd Vdd Inv
XInv_5 inv_W4 W4 Gnd Vdd Inv
XInv_6 inv_W5 W5 Gnd Vdd Inv
XInv_7 inv_W6 W6 Gnd Vdd Inv
XInv_8 inv_W7 W7 Gnd Vdd Inv
XInv_10 inv_W8 W8 Gnd Vdd Inv
XInv_11 inv_W9 W9 Gnd Vdd Inv
XInv_12 inv_W10 W10 Gnd Vdd Inv
XInv_13 inv_W11 W11 Gnd Vdd Inv
XInv_14 inv_W15 W15 Gnd Vdd Inv
XInv_15 inv_W14 W14 Gnd Vdd Inv
```

```
XInv_16 inv_W13 W13 Gnd Vdd Inv
XInv_17 inv_W12 W12 Gnd Vdd Inv
XInv_18 inv_W19 W19 Gnd Vdd Inv
XInv_19 inv_W18 W18 Gnd Vdd Inv
XInv_20 inv_W17 W17 Gnd Vdd Inv
XInv_21 inv_W16 W16 Gnd Vdd Inv
XInv_22 inv_W23 W23 Gnd Vdd Inv
XInv_23 inv_W22 W22 Gnd Vdd Inv
XInv_24 inv_W21 W21 Gnd Vdd Inv
XInv_25 inv_W20 W20 Gnd Vdd Inv
XInv_26 inv_W27 W27 Gnd Vdd Inv
XInv_27 inv_W26 W26 Gnd Vdd Inv
XInv_28 inv_W25 W25 Gnd Vdd Inv
XInv_29 inv_W24 W24 Gnd Vdd Inv
XInv_30 inv_W31 W31 Gnd Vdd Inv
XInv_31 inv_W30 W30 Gnd Vdd Inv
XInv_32 inv_W29 W29 Gnd Vdd Inv
XInv_33 inv_W28 W28 Gnd Vdd Inv
XNAND5_1 inv_A41 inv_A31 inv_A21 inv_A11 inv_A01 inv_W0 Gnd Vdd NAND5
XNAND5_2 inv_A41 inv_A31 inv_A21 inv_A11 A01 inv_W1 Gnd Vdd NAND5
XNAND5_3 inv_A41 inv_A31 inv_A21 A11 inv_A01 inv_W2 Gnd Vdd NAND5
XNAND5_4 inv_A41 inv_A31 inv_A21 A11 A01 inv_W3 Gnd Vdd NAND5
XNAND5_5 inv_A41 inv_A31 A21 inv_A11 inv_A01 inv_W4 Gnd Vdd NAND5
XNAND5_6 inv_A41 inv_A31 A21 inv_A11 A01 inv_W5 Gnd Vdd NAND5
XNAND5_7 inv_A41 inv_A31 A21 A11 inv_A01 inv_W6 Gnd Vdd NAND5
XNAND5_8 inv_A41 inv_A31 A21 A11 A01 inv_W7 Gnd Vdd NAND5
XNAND5_9 inv_A42 A31 inv_A21 inv_A11 inv_A01 inv_W8 Gnd Vdd NAND5
XNAND5_10 inv_A42 A31 inv_A21 inv_A11 A01 inv_W9 Gnd Vdd NAND5
XNAND5_11 inv_A42 A31 inv_A21 A11 inv_A01 inv_W10 Gnd Vdd NAND5
XNAND5_12 inv_A42 A31 inv_A21 A11 A01 inv_W11 Gnd Vdd NAND5
XNAND5_13 inv_A42 A31 A21 inv_A11 inv_A01 inv_W12 Gnd Vdd NAND5
XNAND5_14 inv_A42 A31 A21 inv_A11 A01 inv_W13 Gnd Vdd NAND5
XNAND5_15 inv_A42 A31 A21 A11 inv_A01 inv_W14 Gnd Vdd NAND5
XNAND5_16 inv_A42 A31 A21 A11 A01 inv_W15 Gnd Vdd NAND5
XNAND5_17 A41 inv_A32 inv_A22 inv_A12 inv_A02 inv_W16 Gnd Vdd NAND5
XNAND5_18 A41 inv_A32 inv_A22 inv_A12 A02 inv_W17 Gnd Vdd NAND5
XNAND5_19 A41 inv_A32 inv_A22 A12 inv_A02 inv_W18 Gnd Vdd NAND5
XNAND5_20 A41 inv_A32 inv_A22 A12 A02 inv_W19 Gnd Vdd NAND5
XNAND5_21 A41 inv_A32 A22 inv_A12 inv_A02 inv_W20 Gnd Vdd NAND5
XNAND5_22 A41 inv_A32 A22 inv_A12 A02 inv_W21 Gnd Vdd NAND5
XNAND5_23 A41 inv_A32 A22 A12 inv_A02 inv_W22 Gnd Vdd NAND5
XNAND5_24 A41 inv_A32 A22 A12 A02 inv_W23 Gnd Vdd NAND5
XNAND5_25 A42 A32 inv_A22 inv_A12 inv_A02 inv_W24 Gnd Vdd NAND5
XNAND5_26 A42 A32 inv_A22 inv_A12 A02 inv_W25 Gnd Vdd NAND5
XNAND5_27 A42 A32 inv_A22 A12 inv_A02 inv_W26 Gnd Vdd NAND5
XNAND5_28 A42 A32 inv_A22 A12 A02 inv_W27 Gnd Vdd NAND5
XNAND5_29 A42 A32 A22 inv_A12 inv_A02 inv_W28 Gnd Vdd NAND5
XNAND5_30 A42 A32 A22 inv_A12 A02 inv_W29 Gnd Vdd NAND5
XNAND5_31 A42 A32 A22 A12 inv_A02 inv_W30 Gnd Vdd NAND5
```

```
XNAND5_32 A42 A32 A22 A12 A02 inv_W31 Gnd Vdd NAND5
.ENDS

.SUBCKT N_3 D G S Gnd
M0 D G S Gnd NMOS W='14*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
.ENDS

.SUBCKT P_3 D G S Vdd
M0 D G S Vdd PMOS W='28*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
.ENDS

.SUBCKT LUT_ROM_* SPICE netlist written by S-Edit Win32 6.00
* Written on Nov 19, 1999 at 13:57:23
Vdd Vdd Gnd DC 5
.include "D:\Chris\Thesis\schematics\testfiles\Tapline\2Tap_Test\input_tableTapTest.md"
*
.options  abstol=50n reltol=0.001 numnt=6 prtdel=400n
.tran 300n 68000n start=390n
.print tran "D:\Chris\Thesis\Schematics\testfiles\Tapline\2Tap_Test\CTRLInputs2.out" V(CLK)
V(SL) V(SR) V(LD) V(HLD) V(delta_Phase_inc) V(Phase0) V(Phase1) V(Phase2) V(Phase3)
V(Phase4) V(LD_Phase_inc) V(Tgt_Extent0) V(Tgt_Extent1) V(Range_bin_valid)
V(S_P_Test_Rin) V(S_P_Test_Lin)
.print tran "D:\Chris\Thesis\Schematics\testfiles\Tapline\2Tap_Test\Inputs2.out"
V(Phase_inc0_Tap0) V(Phase_inc1_Tap0) V(Phase_inc2_Tap0) V(Phase_inc3_Tap0)
V(Phase_inc0_Tap1) V(Phase_inc1_Tap1) V(Phase_inc2_Tap1) V(Phase_inc3_Tap1)
V(LD_Gain_Reg) V(Gain_Dat0_Tap0) V(Gain_Dat1_Tap0) V(Gain_Dat0_Tap1)
V(Gain_Dat1_Tap1)
.print tran "D:\Chris\Thesis\Schematics\testfiles\Tapline\2Tap_Test\16Adder1Inputs2.out"
V(Tap_inQ0) V(Tap_inQ1) V(Tap_inQ2) V(Tap_inQ3) V(Tap_inQ4) V(Tap_inQ5)
V(Tap_inQ6) V(Tap_inQ7) V(Tap_inQ8) V(Tap_inQ9) V(Tap_inQ10) V(Tap_inQ11)
V(Tap_inQ12) V(Tap_inQ13) V(Tap_inQ14) V(Tap_inQ15)
.print tran "D:\Chris\Thesis\Schematics\testfiles\Tapline\2Tap_Test\16Adder2Inputs2.out"
V(Tap_inI0) V(Tap_inI1) V(Tap_inI2) V(Tap_inI3) V(Tap_inI4) V(Tap_inI5) V(Tap_inI6)
V(Tap_inI7) V(Tap_inI8) V(Tap_inI9) V(Tap_inI10) V(Tap_inI11) V(Tap_inI12) V(Tap_inI13)
V(Tap_inI14) V(Tap_inI15)
*
.print tran "D:\Chris\Thesis\Schematics\testfiles\Tapline\2Tap_Test\QOUTPUTS2.out"
V(Tap_outQ0)V(Tap_outQ1) V(Tap_outQ2) V(Tap_outQ3) V(Tap_outQ4) V(Tap_outQ5)
V(Tap_outQ6) V(Tap_outQ7) V(Tap_outQ8) V(Tap_outQ9) V(Tap_outQ10) V(Tap_outQ11)
V(Tap_outQ12) V(Tap_outQ13) V(Tap_outQ14) V(Tap_outQ15)
.print tran "D:\Chris\Thesis\Schematics\testfiles\Tapline\2Tap_Test\IOUTPUTS2.out"
V(Tap_outI0)V(Tap_outI1) V(Tap_outI2) V(Tap_outI3) V(Tap_outI4) V(Tap_outI5)
V(Tap_outI6) V(Tap_outI7) V(Tap_outI8) V(Tap_outI9) V(Tap_outI10) V(Tap_outI11)
V(Tap_outI12) V(Tap_outI13) V(Tap_outI14) V(Tap_outI15)
*
.print tran "D:\Chris\Thesis\Schematics\testfiles\Tapline\2Tap_Test\CTRLOutputs2.out"
V(valid_result_out) V(Overflow_out) V(S_P_Test_Lout) V(S_P_Test_Rout)
*
.param l=0.05u
```

```
*
.include "D:\Chris\Thesis\ModelParammod.md"
*
*
* Waveform probing commands
.probe
.options probefilename="D:\Chris\Thesis\Schematics\Testfiles\DISNov16.dat"
+ probesdbfile="D:\Chris\Thesis\Schematics\DISNov16.sdb"
+ probetopmodule="2-Tapline"

* No Ports in cell: PageID_Tanner
* End of module with no ports: PageID_Tanner

.SUBCKT Inv A Out Gnd Vdd
M2 Out A Gnd Gnd NMOS W='28*1' L='7*1' AS='148*1*1' AD='144*1*1' PS='68*1' PD='68*1'
M=1
* Page Size:  5x7
* S-Edit  Inverter (TIB)
* Designed by: J. Luo  Aug 19, 1999  09:04:52
* Schematic generated by S-Edit
* from file D:\Chris\Thesis\Schematics\DISNov16 / module Inv / page Page0
M1 Out A Vdd Vdd PMOS W='28*1' L='7*1' AS='148*1*1' AD='144*1*1' PS='68*1' PD='68*1'
M=1
.ENDS

.SUBCKT NOR2 A B Out Gnd Vdd
M4 Out B Gnd Gnd NMOS W='14*1' L='7*1' AS='144*1*1' AD='84*1*1' PS='68*1' PD='34*1'
M=1
M3 Out A Gnd Gnd NMOS W='14*1' L='7*1' AS='84*1*1' AD='144*1*1' PS='34*1' PD='68*1'
M=1
* Page Size:  5x7
* S-Edit  2-Input NOR Gate (TIB)
* Designed by: J. Luo  Oct 6, 1999  19:12:33
* Schematic generated by S-Edit
* from file D:\Chris\Thesis\Schematics\DISNov16 / module NOR2 / page Page0
M2 Out B 1 Vdd PMOS W='28*1' L='7*1' AS='148*1*1' AD='84*1*1' PS='68*1' PD='34*1' M=1
M1 1 A Vdd Vdd PMOS W='28*1' L='7*1' AS='84*1*1' AD='144*1*1' PS='34*1' PD='68*1' M=1
.ENDS

.SUBCKT OR2 A B Out Gnd Vdd
XInv_1 N4 Out Gnd Vdd Inv
XNOR2_1 A B N4 Gnd Vdd NOR2
.ENDS

.SUBCKT NOR3 A B C Out Gnd Vdd
M4 Out A Gnd Gnd NMOS W='14*1' L='7*1' AS='84*1*1' AD='144*1*1' PS='34*1' PD='68*1'
M=1
M5 Out B Gnd Gnd NMOS W='14*1' L='7*1' AS='84*1*1' AD='84*1*1' PS='34*1' PD='34*1' M=1
```

```
M6 Out C Gnd Gnd NMOS W='14*l' L='7*l' AS='148*l*l' AD='84*l*l' PS='68*l' PD='34*l'
M=1
* Page Size:  5x7
* S-Edit  3-Input NOR Gate (TIB)
* Designed by: J. Luo  Oct 6, 1999  19:13:24
* Schematic generated by S-Edit
* from file D:\Chris\Thesis\Schematics\DISNov16 / module NOR3 / page Page0
M1 1 A Vdd Vdd PMOS W='28*l' L='7*l' AS='84*l*l' AD='144*l*l' PS='34*l' PD='68*l' M=1
M2 2 B 1 Vdd PMOS W='28*l' L='7*l' AS='84*l*l' AD='84*l*l' PS='34*l' PD='34*l' M=1
M3 Out C 2 Vdd PMOS W='28*l' L='7*l' AS='148*l*l' AD='84*l*l' PS='68*l' PD='34*l' M=1
.ENDS

* No Ports in cell: Legend
* End of module with no ports: Legend

.SUBCKT 5-to-32_Decoder_part1 A0 A1 A01 A2 A02 A3 A4 A11 A12 A21 A22 A31 A32
+ A41 A42 inv_A01 inv_A02 inv_A11 inv_A12 inv_A21 inv_A22 inv_A31 inv_A32 inv_A41
+ inv_A42 Gnd Vdd
XInv_1 N2 A42 Gnd Vdd Inv
XInv_2 N2 A41 Gnd Vdd Inv
XInv_3 A4 inv_A42 Gnd Vdd Inv
XInv_4 A4 inv_A41 Gnd Vdd Inv
XInv_5 N3 A32 Gnd Vdd Inv
XInv_6 N3 A31 Gnd Vdd Inv
XInv_7 A3 inv_A32 Gnd Vdd Inv
XInv_8 A3 inv_A31 Gnd Vdd Inv
XInv_9 N9 A22 Gnd Vdd Inv
XInv_10 N9 A21 Gnd Vdd Inv
XInv_11 A2 inv_A22 Gnd Vdd Inv
XInv_12 A2 inv_A21 Gnd Vdd Inv
XInv_13 N15 A12 Gnd Vdd Inv
XInv_14 N15 A11 Gnd Vdd Inv
XInv_15 A1 inv_A12 Gnd Vdd Inv
XInv_16 A1 inv_A11 Gnd Vdd Inv
XInv_17 N19 A01 Gnd Vdd Inv
XInv_18 N19 A02 Gnd Vdd Inv
XInv_19 A0 inv_A02 Gnd Vdd Inv
XInv_20 A0 inv_A01 Gnd Vdd Inv
XInv_21 A4 N2 Gnd Vdd Inv
XInv_22 A3 N3 Gnd Vdd Inv
XInv_23 A1 N15 Gnd Vdd Inv
XInv_24 A2 N9 Gnd Vdd Inv
XInv_25 A0 N19 Gnd Vdd Inv
.ENDS

.SUBCKT NAND5 A B C D E Out Gnd Vdd
M5 Out E N2 Gnd NMOS W='14*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
M6 N2 D N4 Gnd NMOS W='14*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
M7 N4 C N1 Gnd NMOS W='14*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
```

```
M8 N1 B N5 Gnd NMOS W='14*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
M9 N5 A Gnd Gnd NMOS W='14*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
M0 Out E Vdd Vdd PMOS W='28*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
M1 Out D Vdd Vdd PMOS W='28*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
M2 Out C Vdd Vdd PMOS W='28*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
M3 Out B Vdd Vdd PMOS W='28*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
M4 Out A Vdd Vdd PMOS W='28*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
.ENDS

.SUBCKT 5-to-32_Decoder_part2 A01 A02 A11 A12 A21 A22 A31 A32 A41 A42 inv_A01
+ inv_A02 inv_A11 inv_A12 inv_A21 inv_A22 inv_A31 inv_A32 inv_A41 inv_A42 inv_W0
+ inv_W1 inv_W2 inv_W3 inv_W4 inv_W5 inv_W6 inv_W7 inv_W8 inv_W9 inv_W10
inv_W11
+ inv_W12 inv_W13 inv_W14 inv_W15 inv_W16 inv_W17 inv_W18 inv_W19 inv_W20
inv_W21
+ inv_W22 inv_W23 inv_W24 inv_W25 inv_W26 inv_W27 inv_W28 inv_W29 inv_W30
inv_W31
+ W0 W1 W2 W3 W4 W5 W6 W7 W8 W9 W10 W11 W12 W13 W14 W15 W16 W17 W18
W19 W20 W21
+ W22 W23 W24 W25 W26 W27 W28 W29 W30 W31 Gnd Vdd
XInv_1 inv_W1 W1 Gnd Vdd Inv
XInv_2 inv_W0 W0 Gnd Vdd Inv
XInv_3 inv_W2 W2 Gnd Vdd Inv
XInv_4 inv_W3 W3 Gnd Vdd Inv
XInv_5 inv_W4 W4 Gnd Vdd Inv
XInv_6 inv_W5 W5 Gnd Vdd Inv
XInv_7 inv_W6 W6 Gnd Vdd Inv
XInv_8 inv_W7 W7 Gnd Vdd Inv
XInv_10 inv_W8 W8 Gnd Vdd Inv
XInv_11 inv_W9 W9 Gnd Vdd Inv
XInv_12 inv_W10 W10 Gnd Vdd Inv
XInv_13 inv_W11 W11 Gnd Vdd Inv
XInv_14 inv_W15 W15 Gnd Vdd Inv
XInv_15 inv_W14 W14 Gnd Vdd Inv
XInv_16 inv_W13 W13 Gnd Vdd Inv
XInv_17 inv_W12 W12 Gnd Vdd Inv
XInv_18 inv_W19 W19 Gnd Vdd Inv
XInv_19 inv_W18 W18 Gnd Vdd Inv
XInv_20 inv_W17 W17 Gnd Vdd Inv
XInv_21 inv_W16 W16 Gnd Vdd Inv
XInv_22 inv_W23 W23 Gnd Vdd Inv
XInv_23 inv_W22 W22 Gnd Vdd Inv
XInv_24 inv_W21 W21 Gnd Vdd Inv
XInv_25 inv_W20 W20 Gnd Vdd Inv
XInv_26 inv_W27 W27 Gnd Vdd Inv
XInv_27 inv_W26 W26 Gnd Vdd Inv
XInv_28 inv_W25 W25 Gnd Vdd Inv
XInv_29 inv_W24 W24 Gnd Vdd Inv
XInv_30 inv_W31 W31 Gnd Vdd Inv
```

```
XInv_31 inv_W30 W30 Gnd Vdd Inv
XInv_32 inv_W29 W29 Gnd Vdd Inv
XInv_33 inv_W28 W28 Gnd Vdd Inv
XNAND5_1 inv_A41 inv_A31 inv_A21 inv_A11 inv_A01 inv_W0 Gnd Vdd NAND5
XNAND5_2 inv_A41 inv_A31 inv_A21 inv_A11 A01 inv_W1 Gnd Vdd NAND5
XNAND5_3 inv_A41 inv_A31 inv_A21 A11 inv_A01 inv_W2 Gnd Vdd NAND5
XNAND5_4 inv_A41 inv_A31 inv_A21 A11 A01 inv_W3 Gnd Vdd NAND5
XNAND5_5 inv_A41 inv_A31 A21 inv_A11 inv_A01 inv_W4 Gnd Vdd NAND5
XNAND5_6 inv_A41 inv_A31 A21 inv_A11 A01 inv_W5 Gnd Vdd NAND5
XNAND5_7 inv_A41 inv_A31 A21 A11 inv_A01 inv_W6 Gnd Vdd NAND5
XNAND5_8 inv_A41 inv_A31 A21 A11 A01 inv_W7 Gnd Vdd NAND5
XNAND5_9 inv_A42 A31 inv_A21 inv_A11 inv_A01 inv_W8 Gnd Vdd NAND5
XNAND5_10 inv_A42 A31 inv_A21 inv_A11 A01 inv_W9 Gnd Vdd NAND5
XNAND5_11 inv_A42 A31 inv_A21 A11 inv_A01 inv_W10 Gnd Vdd NAND5
XNAND5_12 inv_A42 A31 inv_A21 A11 A01 inv_W11 Gnd Vdd NAND5
XNAND5_13 inv_A42 A31 A21 inv_A11 inv_A01 inv_W12 Gnd Vdd NAND5
XNAND5_14 inv_A42 A31 A21 inv_A11 A01 inv_W13 Gnd Vdd NAND5
XNAND5_15 inv_A42 A31 A21 A11 inv_A01 inv_W14 Gnd Vdd NAND5
XNAND5_16 inv_A42 A31 A21 A11 A01 inv_W15 Gnd Vdd NAND5
XNAND5_17 A41 inv_A32 inv_A22 inv_A12 inv_A02 inv_W16 Gnd Vdd NAND5
XNAND5_18 A41 inv_A32 inv_A22 inv_A12 A02 inv_W17 Gnd Vdd NAND5
XNAND5_19 A41 inv_A32 inv_A22 A12 inv_A02 inv_W18 Gnd Vdd NAND5
XNAND5_20 A41 inv_A32 inv_A22 A12 A02 inv_W19 Gnd Vdd NAND5
XNAND5_21 A41 inv_A32 A22 inv_A12 inv_A02 inv_W20 Gnd Vdd NAND5
XNAND5_22 A41 inv_A32 A22 inv_A12 A02 inv_W21 Gnd Vdd NAND5
XNAND5_23 A41 inv_A32 A22 A12 inv_A02 inv_W22 Gnd Vdd NAND5
XNAND5_24 A41 inv_A32 A22 A12 A02 inv_W23 Gnd Vdd NAND5
XNAND5_25 A42 A32 inv_A22 inv_A12 inv_A02 inv_W24 Gnd Vdd NAND5
XNAND5_26 A42 A32 inv_A22 inv_A12 A02 inv_W25 Gnd Vdd NAND5
XNAND5_27 A42 A32 inv_A22 A12 inv_A02 inv_W26 Gnd Vdd NAND5
XNAND5_28 A42 A32 inv_A22 A12 A02 inv_W27 Gnd Vdd NAND5
XNAND5_29 A42 A32 A22 inv_A12 inv_A02 inv_W28 Gnd Vdd NAND5
XNAND5_30 A42 A32 A22 inv_A12 A02 inv_W29 Gnd Vdd NAND5
XNAND5_31 A42 A32 A22 A12 inv_A02 inv_W30 Gnd Vdd NAND5
XNAND5_32 A42 A32 A22 A12 A02 inv_W31 Gnd Vdd NAND5
.ENDS

.SUBCKT N_3 D G S Gnd
M0 D G S Gnd NMOS W='14*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
.ENDS

.SUBCKT P_3 D G S Vdd
M0 D G S Vdd PMOS W='28*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
.ENDS

.SUBCKT LUT_ROM_* SPICE netlist written by S-Edit Win32 6.00
* Written on Nov 19, 1999 at 13:57:23
Vdd Vdd Gnd DC 5
.include "D:\Chris\Thesis\schematics\testfiles\Tapline\2Tap_Test\input_tableTapTest.md"
```

```
*
.options  abstol=50n reltol=0.001 numnt=6 prtdel=400n
.tran 300n 68000n start=390n
.print tran "D:\Chris\Thesis\Schematics\testfiles\Tapline\2Tap_Test\CTRLInputs2.out" V(CLK)
V(SL) V(SR) V(LD) V(HLD) V(delta_Phase_inc) V(Phase0) V(Phase1) V(Phase2) V(Phase3)
V(Phase4) V(LD_Phase_inc) V(Tgt_Extent0) V(Tgt_Extent1) V(Range_bin_valid)
V(S_P_Test_Rin) V(S_P_Test_Lin)
.print tran "D:\Chris\Thesis\Schematics\testfiles\Tapline\2Tap_Test\Inputs2.out"
V(Phase_inc0_Tap0) V(Phase_inc1_Tap0) V(Phase_inc2_Tap0) V(Phase_inc3_Tap0)
V(Phase_inc0_Tap1) V(Phase_inc1_Tap1) V(Phase_inc2_Tap1) V(Phase_inc3_Tap1)
V(LD_Gain_Reg) V(Gain_Dat0_Tap0) V(Gain_Dat1_Tap0) V(Gain_Dat0_Tap1)
V(Gain_Dat1_Tap1)
.print tran "D:\Chris\Thesis\Schematics\testfiles\Tapline\2Tap_Test\16Adder1Inputs2.out"
V(Tap_inQ0) V(Tap_inQ1) V(Tap_inQ2) V(Tap_inQ3) V(Tap_inQ4) V(Tap_inQ5)
V(Tap_inQ6) V(Tap_inQ7) V(Tap_inQ8) V(Tap_inQ9) V(Tap_inQ10) V(Tap_inQ11)
V(Tap_inQ12) V(Tap_inQ13) V(Tap_inQ14) V(Tap_inQ15)
.print tran "D:\Chris\Thesis\Schematics\testfiles\Tapline\2Tap_Test\16Adder2Inputs2.out"
V(Tap_inI0) V(Tap_inI1) V(Tap_inI2) V(Tap_inI3) V(Tap_inI4) V(Tap_inI5) V(Tap_inI6)
V(Tap_inI7) V(Tap_inI8) V(Tap_inI9) V(Tap_inI10) V(Tap_inI11) V(Tap_inI12) V(Tap_inI13)
V(Tap_inI14) V(Tap_inI15)
*
.print tran "D:\Chris\Thesis\Schematics\testfiles\Tapline\2Tap_Test\QOUTPUTS2.out"
V(Tap_outQ0)V(Tap_outQ1) V(Tap_outQ2) V(Tap_outQ3) V(Tap_outQ4) V(Tap_outQ5)
V(Tap_outQ6) V(Tap_outQ7) V(Tap_outQ8) V(Tap_outQ9) V(Tap_outQ10) V(Tap_outQ11)
V(Tap_outQ12) V(Tap_outQ13) V(Tap_outQ14) V(Tap_outQ15)
.print tran "D:\Chris\Thesis\Schematics\testfiles\Tapline\2Tap_Test\IOUTPUTS2.out"
V(Tap_outI0)V(Tap_outI1) V(Tap_outI2) V(Tap_outI3) V(Tap_outI4) V(Tap_outI5)
V(Tap_outI6) V(Tap_outI7) V(Tap_outI8) V(Tap_outI9) V(Tap_outI10) V(Tap_outI11)
V(Tap_outI12) V(Tap_outI13) V(Tap_outI14) V(Tap_outI15)
*
.print tran "D:\Chris\Thesis\Schematics\testfiles\Tapline\2Tap_Test\CTRLOutputs2.out"
V(valid_result_out) V(Overflow_out) V(S_P_Test_Lout) V(S_P_Test_Rout)
*
.param l=0.05u
*
.include "D:\Chris\Thesis\ModelParammod.md"
*
*
* Waveform probing commands
.probe
.options probefilename="D:\Chris\Thesis\Schematics\Testfiles\DISNov16.dat"
+ probesdbfile="D:\Chris\Thesis\Schematics\DISNov16.sdb"
+ probetopmodule="2-Tapline"

* No Ports in cell: PageID_Tanner
* End of module with no ports: PageID_Tanner

.SUBCKT Inv A Out Gnd Vdd
```

```
M2 Out A Gnd Gnd NMOS W='28*l' L='7*l' AS='148*l*l' AD='144*l*l' PS='68*l' PD='68*l'
M=1
* Page Size: 5x7
* S-Edit Inverter (TIB)
* Designed by: J. Luo Aug 19, 1999 09:04:52
* Schematic generated by S-Edit
* from file D:\Chris\Thesis\Schematics\DISNov16 / module Inv / page Page0
M1 Out A Vdd Vdd PMOS W='28*l' L='7*l' AS='148*l*l' AD='144*l*l' PS='68*l' PD='68*l'
M=1
.ENDS

.SUBCKT NOR2 A B Out Gnd Vdd
M4 Out B Gnd Gnd NMOS W='14*l' L='7*l' AS='144*l*l' AD='84*l*l' PS='68*l' PD='34*l'
M=1
M3 Out A Gnd Gnd NMOS W='14*l' L='7*l' AS='84*l*l' AD='144*l*l' PS='34*l' PD='68*l'
M=1
* Page Size: 5x7
* S-Edit 2-Input NOR Gate (TIB)
* Designed by: J. Luo Oct 6, 1999 19:12:33
* Schematic generated by S-Edit
* from file D:\Chris\Thesis\Schematics\DISNov16 / module NOR2 / page Page0
M2 Out B 1 Vdd PMOS W='28*l' L='7*l' AS='148*l*l' AD='84*l*l' PS='68*l' PD='34*l' M=1
M1 1 A Vdd Vdd PMOS W='28*l' L='7*l' AS='84*l*l' AD='144*l*l' PS='34*l' PD='68*l' M=1
.ENDS

.SUBCKT OR2 A B Out Gnd Vdd
XInv_1 N4 Out Gnd Vdd Inv
XNOR2_1 A B N4 Gnd Vdd NOR2
.ENDS

.SUBCKT NOR3 A B C Out Gnd Vdd
M4 Out A Gnd Gnd NMOS W='14*l' L='7*l' AS='84*l*l' AD='144*l*l' PS='34*l' PD='68*l'
M=1
M5 Out B Gnd Gnd NMOS W='14*l' L='7*l' AS='84*l*l' AD='84*l*l' PS='34*l' PD='34*l' M=1
M6 Out C Gnd Gnd NMOS W='14*l' L='7*l' AS='148*l*l' AD='84*l*l' PS='68*l' PD='34*l'
M=1
* Page Size: 5x7
* S-Edit 3-Input NOR Gate (TIB)
* Designed by: J. Luo Oct 6, 1999 19:13:24
* Schematic generated by S-Edit
* from file D:\Chris\Thesis\Schematics\DISNov16 / module NOR3 / page Page0
M1 1 A Vdd Vdd PMOS W='28*l' L='7*l' AS='84*l*l' AD='144*l*l' PS='34*l' PD='68*l' M=1
M2 2 B 1 Vdd PMOS W='28*l' L='7*l' AS='84*l*l' AD='84*l*l' PS='34*l' PD='34*l' M=1
M3 Out C 2 Vdd PMOS W='28*l' L='7*l' AS='148*l*l' AD='84*l*l' PS='68*l' PD='34*l' M=1
.ENDS

* No Ports in cell: Legend
* End of module with no ports: Legend
```

```
.SUBCKT 5-to-32_Decoder_part1 A0 A1 A01 A2 A02 A3 A4 A11 A12 A21 A22 A31 A32
+ A41 A42 inv_A01 inv_A02 inv_A11 inv_A12 inv_A21 inv_A22 inv_A31 inv_A32 inv_A41
+ inv_A42 Gnd Vdd
XInv_1 N2 A42 Gnd Vdd Inv
XInv_2 N2 A41 Gnd Vdd Inv
XInv_3 A4 inv_A42 Gnd Vdd Inv
XInv_4 A4 inv_A41 Gnd Vdd Inv
XInv_5 N3 A32 Gnd Vdd Inv
XInv_6 N3 A31 Gnd Vdd Inv
XInv_7 A3 inv_A32 Gnd Vdd Inv
XInv_8 A3 inv_A31 Gnd Vdd Inv
XInv_9 N9 A22 Gnd Vdd Inv
XInv_10 N9 A21 Gnd Vdd Inv
XInv_11 A2 inv_A22 Gnd Vdd Inv
XInv_12 A2 inv_A21 Gnd Vdd Inv
XInv_13 N15 A12 Gnd Vdd Inv
XInv_14 N15 A11 Gnd Vdd Inv
XInv_15 A1 inv_A12 Gnd Vdd Inv
XInv_16 A1 inv_A11 Gnd Vdd Inv
XInv_17 N19 A01 Gnd Vdd Inv
XInv_18 N19 A02 Gnd Vdd Inv
XInv_19 A0 inv_A02 Gnd Vdd Inv
XInv_20 A0 inv_A01 Gnd Vdd Inv
XInv_21 A4 N2 Gnd Vdd Inv
XInv_22 A3 N3 Gnd Vdd Inv
XInv_23 A1 N15 Gnd Vdd Inv
XInv_24 A2 N9 Gnd Vdd Inv
XInv_25 A0 N19 Gnd Vdd Inv
.ENDS

.SUBCKT NAND5 A B C D E Out Gnd Vdd
M5 Out E N2 Gnd NMOS W='14*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
M6 N2 D N4 Gnd NMOS W='14*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
M7 N4 C N1 Gnd NMOS W='14*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
M8 N1 B N5 Gnd NMOS W='14*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
M9 N5 A Gnd Gnd NMOS W='14*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
M0 Out E Vdd Vdd PMOS W='28*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
M1 Out D Vdd Vdd PMOS W='28*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
M2 Out C Vdd Vdd PMOS W='28*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
M3 Out B Vdd Vdd PMOS W='28*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
M4 Out A Vdd Vdd PMOS W='28*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
.ENDS

.SUBCKT 5-to-32_Decoder_part2 A01 A02 A11 A12 A21 A22 A31 A32 A41 A42 inv_A01
+ inv_A02 inv_A11 inv_A12 inv_A21 inv_A22 inv_A31 inv_A32 inv_A41 inv_A42 inv_W0
+ inv_W1 inv_W2 inv_W3 inv_W4 inv_W5 inv_W6 inv_W7 inv_W8 inv_W9 inv_W10
inv_W11
+ inv_W12 inv_W13 inv_W14 inv_W15 inv_W16 inv_W17 inv_W18 inv_W19 inv_W20
inv_W21
```

```
+ inv_W22 inv_W23 inv_W24 inv_W25 inv_W26 inv_W27 inv_W28 inv_W29 inv_W30
inv_W31
+ W0 W1 W2 W3 W4 W5 W6 W7 W8 W9 W10 W11 W12 W13 W14 W15 W16 W17 W18
W19 W20 W21
+ W22 W23 W24 W25 W26 W27 W28 W29 W30 W31 Gnd Vdd
XInv_1 inv_W1 W1 Gnd Vdd Inv
XInv_2 inv_W0 W0 Gnd Vdd Inv
XInv_3 inv_W2 W2 Gnd Vdd Inv
XInv_4 inv_W3 W3 Gnd Vdd Inv
XInv_5 inv_W4 W4 Gnd Vdd Inv
XInv_6 inv_W5 W5 Gnd Vdd Inv
XInv_7 inv_W6 W6 Gnd Vdd Inv
XInv_8 inv_W7 W7 Gnd Vdd Inv
XInv_10 inv_W8 W8 Gnd Vdd Inv
XInv_11 inv_W9 W9 Gnd Vdd Inv
XInv_12 inv_W10 W10 Gnd Vdd Inv
XInv_13 inv_W11 W11 Gnd Vdd Inv
XInv_14 inv_W15 W15 Gnd Vdd Inv
XInv_15 inv_W14 W14 Gnd Vdd Inv
XInv_16 inv_W13 W13 Gnd Vdd Inv
XInv_17 inv_W12 W12 Gnd Vdd Inv
XInv_18 inv_W19 W19 Gnd Vdd Inv
XInv_19 inv_W18 W18 Gnd Vdd Inv
XInv_20 inv_W17 W17 Gnd Vdd Inv
XInv_21 inv_W16 W16 Gnd Vdd Inv
XInv_22 inv_W23 W23 Gnd Vdd Inv
XInv_23 inv_W22 W22 Gnd Vdd Inv
XInv_24 inv_W21 W21 Gnd Vdd Inv
XInv_25 inv_W20 W20 Gnd Vdd Inv
XInv_26 inv_W27 W27 Gnd Vdd Inv
XInv_27 inv_W26 W26 Gnd Vdd Inv
XInv_28 inv_W25 W25 Gnd Vdd Inv
XInv_29 inv_W24 W24 Gnd Vdd Inv
XInv_30 inv_W31 W31 Gnd Vdd Inv
XInv_31 inv_W30 W30 Gnd Vdd Inv
XInv_32 inv_W29 W29 Gnd Vdd Inv
XInv_33 inv_W28 W28 Gnd Vdd Inv
XNAND5_1 inv_A41 inv_A31 inv_A21 inv_A11 inv_A01 inv_W0 Gnd Vdd NAND5
XNAND5_2 inv_A41 inv_A31 inv_A21 inv_A11 A01 inv_W1 Gnd Vdd NAND5
XNAND5_3 inv_A41 inv_A31 inv_A21 A11 inv_A01 inv_W2 Gnd Vdd NAND5
XNAND5_4 inv_A41 inv_A31 inv_A21 A11 A01 inv_W3 Gnd Vdd NAND5
XNAND5_5 inv_A41 inv_A31 A21 inv_A11 inv_A01 inv_W4 Gnd Vdd NAND5
XNAND5_6 inv_A41 inv_A31 A21 inv_A11 A01 inv_W5 Gnd Vdd NAND5
XNAND5_7 inv_A41 inv_A31 A21 A11 inv_A01 inv_W6 Gnd Vdd NAND5
XNAND5_8 inv_A41 inv_A31 A21 A11 A01 inv_W7 Gnd Vdd NAND5
XNAND5_9 inv_A42 A31 inv_A21 inv_A11 inv_A01 inv_W8 Gnd Vdd NAND5
XNAND5_10 inv_A42 A31 inv_A21 inv_A11 A01 inv_W9 Gnd Vdd NAND5
XNAND5_11 inv_A42 A31 inv_A21 A11 inv_A01 inv_W10 Gnd Vdd NAND5
XNAND5_12 inv_A42 A31 inv_A21 A11 A01 inv_W11 Gnd Vdd NAND5
```

```
XNAND5_13 inv_A42 A31 A21 inv_A11 inv_A01 inv_W12 Gnd Vdd NAND5
XNAND5_14 inv_A42 A31 A21 inv_A11 A01 inv_W13 Gnd Vdd NAND5
XNAND5_15 inv_A42 A31 A21 A11 inv_A01 inv_W14 Gnd Vdd NAND5
XNAND5_16 inv_A42 A31 A21 A11 A01 inv_W15 Gnd Vdd NAND5
XNAND5_17 A41 inv_A32 inv_A22 inv_A12 inv_A02 inv_W16 Gnd Vdd NAND5
XNAND5_18 A41 inv_A32 inv_A22 inv_A12 A02 inv_W17 Gnd Vdd NAND5
XNAND5_19 A41 inv_A32 inv_A22 A12 inv_A02 inv_W18 Gnd Vdd NAND5
XNAND5_20 A41 inv_A32 inv_A22 A12 A02 inv_W19 Gnd Vdd NAND5
XNAND5_21 A41 inv_A32 A22 inv_A12 inv_A02 inv_W20 Gnd Vdd NAND5
XNAND5_22 A41 inv_A32 A22 inv_A12 A02 inv_W21 Gnd Vdd NAND5
XNAND5_23 A41 inv_A32 A22 A12 inv_A02 inv_W22 Gnd Vdd NAND5
XNAND5_24 A41 inv_A32 A22 A12 A02 inv_W23 Gnd Vdd NAND5
XNAND5_25 A42 A32 inv_A22 inv_A12 inv_A02 inv_W24 Gnd Vdd NAND5
XNAND5_26 A42 A32 inv_A22 inv_A12 A02 inv_W25 Gnd Vdd NAND5
XNAND5_27 A42 A32 inv_A22 A12 inv_A02 inv_W26 Gnd Vdd NAND5
XNAND5_28 A42 A32 inv_A22 A12 A02 inv_W27 Gnd Vdd NAND5
XNAND5_29 A42 A32 A22 inv_A12 inv_A02 inv_W28 Gnd Vdd NAND5
XNAND5_30 A42 A32 A22 inv_A12 A02 inv_W29 Gnd Vdd NAND5
XNAND5_31 A42 A32 A22 A12 inv_A02 inv_W30 Gnd Vdd NAND5
XNAND5_32 A42 A32 A22 A12 A02 inv_W31 Gnd Vdd NAND5
.ENDS

.SUBCKT N_3 D G S Gnd
M0 D G S Gnd NMOS W='14*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
.ENDS

.SUBCKT P_3 D G S Vdd
M0 D G S Vdd PMOS W='28*l' L='7*l' AS='66*l*l' AD='66*l*l' PS='24*l' PD='24*l' M=1
.ENDS

.SUBCKT LUT_ROM_* SPICE netlist written by S-Edit Win32 6.00
* Written on Nov 19, 1999 at 13:57:23
Vdd Vdd Gnd DC 5
.include "D:\Chris\Thesis\schematics\testfiles\Tapline\2Tap_Test\input_tableTapTest.md"
*
.options  abstol=50n reltol=0.001 numnt=6 prtdel=400n
.tran 300n 68000n start=390n
.print tran "D:\Chris\Thesis\Schematics\testfiles\Tapline\2Tap_Test\CTRLInputs2.out" V(CLK)
V(SL) V(SR) V(LD) V(HLD) V(delta_Phase_inc) V(Phase0) V(Phase1) V(Phase2) V(Phase3)
V(Phase4) V(LD_Phase_inc) V(Tgt_Extent0) V(Tgt_Extent1) V(Range_bin_valid)
V(S_P_Test_Rin) V(S_P_Test_Lin)
.print tran "D:\Chris\Thesis\Schematics\testfiles\Tapline\2Tap_Test\Inputs2.out"
V(Phase_inc0_Tap0) V(Phase_inc1_Tap0) V(Phase_inc2_Tap0) V(Phase_inc3_Tap0)
V(Phase_inc0_Tap1) V(Phase_inc1_Tap1) V(Phase_inc2_Tap1) V(Phase_inc3_Tap1)
V(LD_Gain_Reg) V(Gain_Dat0_Tap0) V(Gain_Dat1_Tap0) V(Gain_Dat0_Tap1)
V(Gain_Dat1_Tap1)
.print tran "D:\Chris\Thesis\Schematics\testfiles\Tapline\2Tap_Test\16Adder1Inputs2.out"
V(Tap_inQ0) V(Tap_inQ1) V(Tap_inQ2) V(Tap_inQ3) V(Tap_inQ4) V(Tap_inQ5)
```

V(Tap_inQ6) V(Tap_inQ7) V(Tap_inQ8) V(Tap_inQ9) V(Tap_inQ10) V(Tap_inQ11)
V(Tap_inQ12) V(Tap_inQ13) V(Tap_inQ14) V(Tap_inQ15)
.print tran "D:\Chris\Thesis\Schematics\testfiles\Tapline\2Tap_Test\16Adder2I

Hardlimiter m-file

```
% hard_limiter2Taps.m
%*********************************************************************
% Test file...
% - reads modified text files generated from T-Spice output files
% - extraxts and assigns values to variables
% - hard limiters values into binary representation
% - writes results in decimal form to text files
%
% Created by: MAJ Stig Ekestorm, Nov-99
% Naval Postgraduate School
%*********************************************************************


% *** READ IN VALUES FROM T-SPICE OUTPUT FILES (in text file format) ***

clear           %clear all variables

%for the 2-Tap simulation
row=160;        %# of rows
%colin=6;       %# of columns for input
colout=17;      %# of columns for output

%open text files to read values
%fin = fopen('     .txt','r');
foutI = fopen('IOutputs2_MOD.txt','r');
foutQ = fopen('QOutputs2_MOD.txt','r');

%initalize
%tmpin=zeros(row,colin);
tmpoutI=zeros(row,colout);
tmpoutQ=zeros(row,colout);

%extract values from the text file
%for idx2=1:row,                                    %# of rows
%   for idx3=1:colin,                               %# of columns
%      tmpin(idx2,idx3)=fscanf(fin,'%f',1);         %reads in the values
%   end
%end
for idx2=1:row,                                     %# of rows of valid result
out
   for idx3=1:colout,                               %# of columns, OBS: 1st
column is time
      tmpoutI(idx2,idx3)=fscanf(foutI,'%f',1);      %reads in the values
      tmpoutQ(idx2,idx3)=fscanf(foutQ,'%f',1);  %reads in the values
   end
end

%close text files
%fclose(fin);
fclose(foutI);
%fclose(foutQ);
```

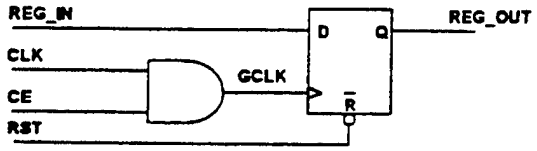
```
% *** EXTRACT/SEPARATE VARIABLES ***

%initalize
time=zeros(row,1);
%in=zeros(row,colin-1);
outI=zeros(row,colout-1);
outQ=zeros(row,colout-1);
%input=zeros(row,colin-1);
Iout=zeros(row,colout-1);
Qout=zeros(row,colout-1);

%assign values to correct variable
time=tmpoutI(:,1);
%in=tmpin(:,2:colin);
outI=tmpoutI(:,2:colout);
outQ=tmpoutQ(:,2:colout);


% *** HARD LIMITER ***

%hard limiter
for idx4=1:row,
   %for idx5=1:colin-1,
   %   if in(idx4,idx5)<=2.5,     %if less then 2.5V
   %     input(idx4,idx5)=0;      %set bit to "0"
   %  else                        %if higher then 2.5V
   %     input(idx4,idx5)=1;      %set bit to "1"
   %   end
   %end
   for idx5=1:colout-1,
      if outI(idx4,idx5)<=2.5,    %if less then 2.5V
         Iout(idx4,idx5)=0;       %set bit to "0"
      else                        %if higher then 2.5V
         Iout(idx4,idx5)=1;       %set bit to "1"
      end
   end
   for idx5=1:colout-1,
      if outQ(idx4,idx5)<=2.5,    %if less then 2.5V
         Qout(idx4,idx5)=0;       %set bit to "0"
      else                        %if higher then 2.5V
         Qout(idx4,idx5)=1;       %set bit to "1"
      end
   end
end

%flip matrices to get MSB to the left, and LSB to the right
%input=fliplr(input);
Iout=fliplr(Iout);
Qout=fliplr(Qout);
```

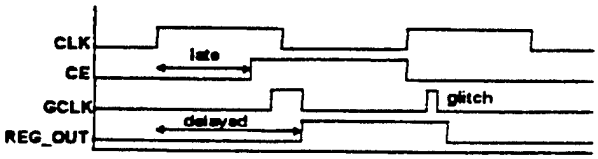
```
% *** PRINT TO MATLAB COMMAND WINDOW ***

%print input and output matrices to MATLAB Command Window
%disp(' ')
%disp('Input vectors:')
%input
%disp(' ')
%disp('Output vectors:')
%Iout
%Qout
%disp(' ')


% *** PRINT TO TEXT FILES ***

%print I- and Q-output in decimal values to two separate text files
f1=fopen('imagei2_2Taps.txt','w');      %open text file to write results to
f2=fopen('imageq2_2Taps.txt','w');      %open text file to write results to
fprintf(f1,'%d\n',two2dec(Iout,7));     %write I-values as decimal value
fprintf(f2,'%d\n',two2dec(Qout,7));     %write Q-values as decimal value
fclose(f1);                             %close text file that results has been
written to
fclose(f2);                             %close text file that results has been
written to


%end of file
```

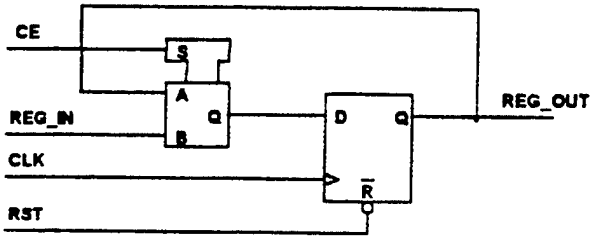
```
%compare_2Taps.m

% *** PRINT TO TEXT FILES ***

%print I- and Q-output in decimal values to two separate text files
f1=fopen('imagei2_2TapsMATLAB.txt','w');     %open text file to write results
to
f2=fopen('imageq2_2TapsMATLAB.txt','w');     %open text file to write results
to

for idx1=1:10,
   for idx2=1:16,
      fprintf(f1,'%d\n',real(finalAdderOut(idx1,idx2)));    %write I-values
      fprintf(f2,'%d\n',imag(finalAdderOut(idx1,idx2)));    %write Q-values
   end
end

fclose(f1);                                %close text file that results has been
written to
fclose(f2);                                %close text file that results has been
written to


% *** COMPARE RESULTS ***

load -ascii imagei2_2Taps.txt
load -ascii imageq2_2Taps.txt
load -ascii imagei2_2TapsMATLAB.txt
load -ascii imageq2_2TapsMATLAB.txt

DiffI=imagei2_2Taps-imagei2_2TapsMATLAB;
DiffQ=imageq2_2Taps-imageq2_2TapsMATLAB;

Difference_I=find(DiffI)
Difference_Q=find(DiffI)


%end of file
```

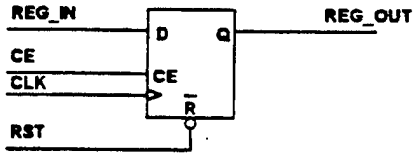
Appendix E - AMI

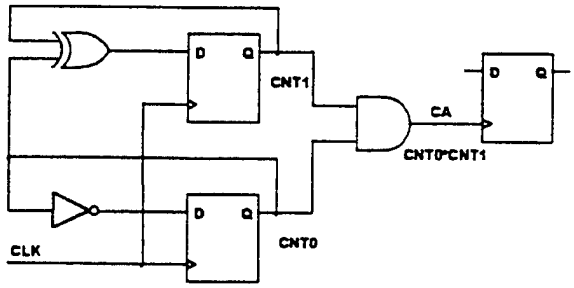
These information are based on an American Microsystems Inc. (AMI) presentation during a seminar for ASIC technology. The following table should considered as a source of design improvement before conducting the process of conversion from an FPGA design to an ASIC layout. To simplify the table, it will be limited to a schematic diagram, a description of problems that may occur for a non-optimum layout and finally the improvements for an optimum layout.

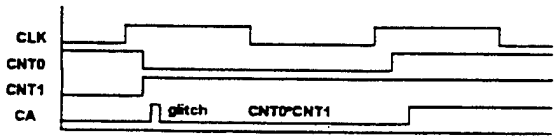
<table>
<tr><th>Layout to avoid</th><th>Optimized Layout for ASIC</th></tr>
<tr><td>
Gated Clocks
• asynchronous • low power shut-down
• glitch sensitive • test problem
• fast data path


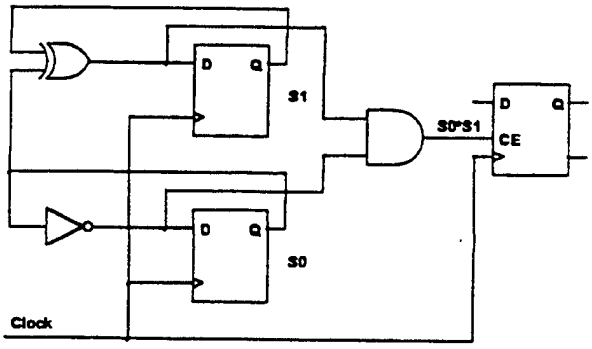


</td><td>
Clock-Enable Flip-Flop
• synchronous • uses FPGA CE resources
• glitch insensitive • data path MUX slower


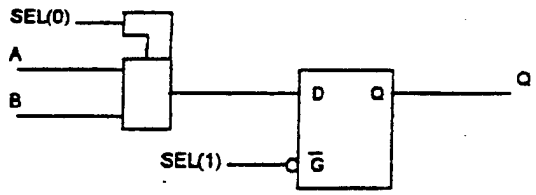
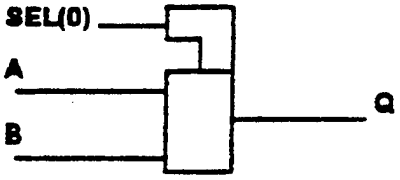
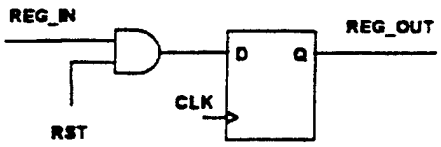
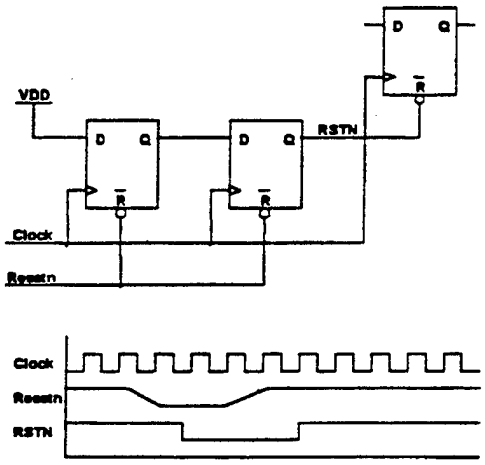


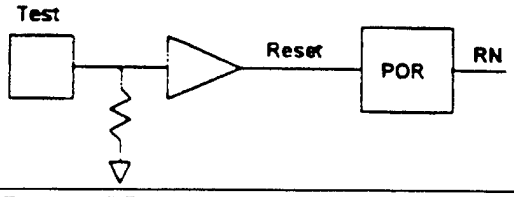
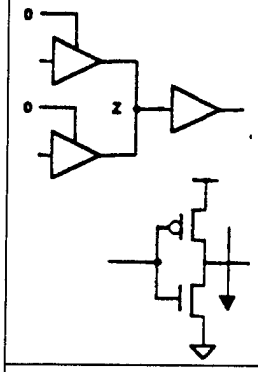
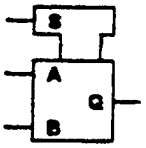
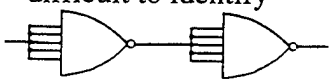
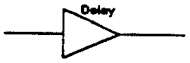
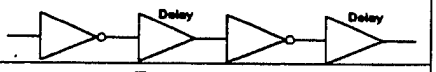
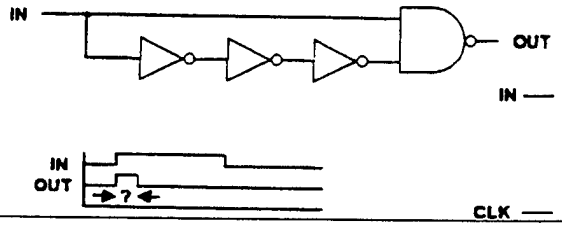
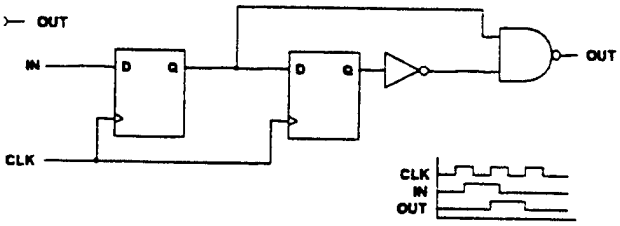
</td></tr>
<tr><td>
Glitching State Decoder
• technology dependant • test problem




</td><td>
Synronous State Decoder
• no latency • fully synchronous


</td></tr>
</table>

Layout to avoid	Optimized Layout for ASIC
Latch Inference: CASE Statement (VHDL/Verilog) • unintended latch • partial case condition coverage • resources wasted 	**Latch Inference** Avoided • default clause covers all conditions • efficient implementation • don't cares optimize implementation 
Synchronous Reset • requires free-running clock • affects performance 	**Asynchronous Reset** • clock independent • simplifies simulation • insures Finite State Machines (FSM) in known state before first clock • avoids FSM latchup (dead) states • minimal impact on performance • direct synthesis to library flip-flop
Asynchronous Set-Reset • set-reset priority makes a difference • behavioral and structural simulation may mismatch • violates definition of synchronous design (only Set or Reset but not both) • many FPGAs don't support Set and Reset	**Synchronized Reset** • asynchronous reset • synchronized recovery 

Layout to avoid	Optimized Layout for ASIC
	Advantages for Power-On-Resets • timing characteristics • voltage sensivity • testability issues • synchronizing multiple PORs • avoid using internal PORs • use single external POR generator 
Internal Tri-State (unclear) • not sure what to check for 	**Internal Tri-State** (clear) • iddq testability issue • bus contention • use pull-up/down or bus latch • muxes are better • automatic conversion tools 
Bad use of Logic Gates as Delays • unpredictable • technology dependant • difficult to identify 	**Good use of Logic Gates as Delays** • easier to identify  • symmetrical rise/fall 
Bad Pulse Generators Layout • unpredictable • technology dependant • test problem 	**Good Pulse Generators Layout** • technology independent 

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Rd, STE. 0944
   Alexandria, Virginia 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, California 93943-5101

3. Research Office, Code 09.....1
   Naval Postgraduate School
   589 Dyer Rd.
   Monterey, CA 93943-5138

4. Chairman, Code EC.....1
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   833 Dyer Rd.
   Monterey, CA 93943-5121

5. Dr. Phillip Pace, Code EC/PC 1
   Department of Electrical and Computing Engineering
   Naval Postgraduate School
   Monterey, California 93943-5121

6. Commanding Officer Naval Research Laboratory 1
   Attn: Dr. John Montgomery
   Code 5700.00
   4555 Overlook Avenue, S.W.
   Washington, D.C. 20375-5339

7. Commanding Officer Naval Research Laboratory 1
   Attn: Mr. Alfred DiMattesa
   Code 5701.00
   4555 Overlook Avenue, S.W.
   Washington, D.C. 20375-5339

8. Commanding Officer Naval Research Laboratory 1
   Attn: Dr. Joseph Lawrence
   Code 5740.00
   4555 Overlook Avenue, S.W.
   Washington, D.C. 20375-5339

9. Commanding Officer Naval Research Laboratory 1
   Attn: Mr. Gregory Hrin
   Code 5742.00
   4555 Overlook Avenue, S.W.
   Washington, D.C. 20375-5339

10. Commanding Officer Naval Research Laboratory 1
    Attn: Mr. Dan Bay
    Code 5742.01
    4555 Overlook Avenue, S.W.
    Washington, D.C. 20375-5339

11. Commanding Officer Naval Research Laboratory 1
    Attn: Mr. Jon Uffelman
    Code 5740.00
    4555 Overlook Avenue, S.W.
    Washington, D.C. 20375-5339

12. Commanding Officer Naval Research Laboratory 1
    Attn: Mr. Brian Edwards
    Code 5760.00
    4555 Overlook Avenue, S.W.
    Washington, D.C. 20375-5339

13. Commanding Officer Naval Research Laboratory 1
    Attn: Mr. Robert E. Surratt
    Code 5760.00
    4555 Overlook Avenue, S.W.
    Washington, D.C. 20375-5339

14. Commanding Officer Office of Naval Research 1
    Attn: Dr. Harry Hurt
    Code ONR-313EW
    800 North Quincy Street
    Arlington VA. 22217-5660